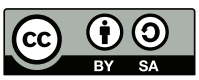
Text Analysis and Retrieval 2018

Course Project Reports

University of Zagreb

Faculty of Electrical Engineering and Computing



Publisher:

University of Zagreb, Faculty of Electrical Engineering and Computing

Organizers:

Mladen Karan
Domagoj Alagić
Jan Šnajder

Editors:

Mladen Karan
Domagoj Alagić
Jan Šnajder

ISBN 978-953-184-251-8

Course website:

http://www.fer.unizg.hr/predmet/apt

Powered by:

Text Analysis and Knowledge Engineering Lab
University of Zagreb
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
http://takelab.fer.hr

TakeLab

Preface

This is the fifth booklet in a series of students' project reports from the Text Analysis and Retrieval (TAR) course, taught at the Faculty of Electrical Engineering and Computing (FER), University of Zagreb. TAR teaches the foundations of natural language processing and information retrieval methods, with a particular attention given to their practical applications, all while being in line with the best practices in the area.

This is achieved through hands-on student projects, which are the central part of the course. Students that complete this course gain both practical and theoretical skills in data science and machine learning, with a strong focus on text data. Given the bewildering and ever-growing amount of information available today (in text form especially), such skills are invaluable to employers. We are happy and proud to supply our students with such an extra edge on the increasingly competitive job market. This booklet represents the results of 24 projects, which are the work of 63 students. Most of the topics were adopted from recent workshops and shared tasks like SemEval and CLEF. However, some of the students proposed their own, which resulted in a very diverse set of topics, including toxic language detection, emotion analysis, spam classification, knowledge extraction from medical texts, depression detection, and question answering, to name a few. With respect to methods, while many teams relied on traditional machine learning methods, a significant number of teams also ventured into (considerably advanced) deep learning models. This is not so surprising, as recently there has been a clear trend of deep learning approaches gradually taking over the spotlight.

As in the previous years, the project reports were written in the form of a research paper. The purpose of this was to both teach the students to effectively present their results, as well as to give them a feeling of how actual scientific research works. To this end, in addition to writing a project report, the students also actively participated in several paper reading sessions, where scientific papers were discussed in groups. As students seldom get an opportunity to get involved in hands-on scientific research during their Master's programme, we felt this would be a valuable new experience for them. And we weren't wrong: this was confirmed by resoundingly positive student feedback.

As the course organizers, we thank the students for demonstrating remarkable motivation and enthusiasm throughout this course. We are honored to have had the opportunity to work with them.

Mladen Karan, Domagoj Alagić, and Jan Šnajder

Contents

Character Identification on Multiparty Dialogues Using Mention-Pair Coreference Resolution

Dan Ambrošić, Stjepan Dugonjić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{dan.ambrosic,stjepan.dugonjic}@fer.hr

Abstract

This paper describes a solution to the SemEval 2018 task 4 whose problem is character identification on multiparty dialogues from the TV show *Friends*. The problem is split into coreference resolution and entity linking. Our solution uses a mention-pair coreference model and we trained a binary random forest classifier, on a set of simple features, for predicting if a pair of mentions is coreferent or not. That in conjunction with a few entity linking rules yielded competitive results on the official test data.

1. Introduction

Character identification is essentially an entity linking task with the goal of identifying each mention as a certain character from the knowledge-base of characters, that is, to assign each mention to its entity. In this work we used the transcripts form the popular television show *Friends*. For example, if Chandler says: "Honey, I love you.", the system should map the words "Honey" and "you" to Monica. A mention could also refer to a character that is not present in the current conversation, making the identification more difficult. A short conversation from Figure 1 illustrates that scenario, with Jack and Judy not being present. This requires the system to handle cross-document entity resolution.

First step of character identification is mention detection which was already done on the dataset we used. After all character mentions are annotated, we could train a classifier to predict which character is each mention refering to. Unfortunately, that is a hard problem since there are potentially a lot of characters, most of whom appear in just a few scenes, and the classifier would need to get a lot of contextual information. In order to make the entity linking task easier, we need to apply coreference resolution that will generate coreference chains. Those chains group together mentions referring to the same character and if we can infer which character at least one mention refers to, we can map the whole chain to that character.

2. Related work

There exist different approaches for tackling coreference resolution. The most basic one is a deterministic approach elaborated in Lee et al. (2011) which uses lexical, syntactic, semantic and discourse information to develop heuristic rules. With popularization of machine learning, more complex models have been used to improve the results. Soon et al. (2001) were the first to achieve results comparable to deterministic approaches using a supervised mention-pair model which was based on a decision tree classifier. Mention-pair models predict if a pair of mentions is coreferent or not and are explained in more detail in section 4.1.

In recent years, advancements in deep learning offered improvements to such models and Clark and Manning (2016) developed a mention-ranking model using deep reinforcement learning. Besides those, Wiseman et al. (2016) have used recurrent neural networks to learn latent, global representations of entity clusters directly from their mentions and achieved state-of-the-art performance.

The task of character identification on TV show transcripts was already explored by Chen and Choi (2016). For coreference resolution, they used the aforementioned deterministic system, as well as an entity-centric model developed in Clark and Manning (2015), and achieved admirable results. Since most of the work on entity linking was focused on Wikification, they proposed a set of rules for mapping each coreference chain to one character. The same authors later presented a robust system for character identification based on an agglomerative convolutional neural network in Chen et al. (2017).

Our work combines previous methods. Instead of one decision tree, we trained a supervised random forest mention-pair classifier used for generating coreference chains. Feature extraction was done with regard to the generic features proposed by Bengtson and Roth (2008). However, the number of features we used is smaller since this task only deals with mentions of persons. After the model is trained, coreference chains are built on the test data and each one is then mapped to a character using the same set of rules as Chen and Choi (2016) used.

3. Dataset

We used the data provided for SemEval 2018 Task 4[1] which is split into a training and a test set, both of which are formatted per CoNLL-2012 Shared Task standard. They contain transcripts of the first two seasons of the TV show *Friends*. Each season is split into episodes, each episode into scenes, each scene into sentences and each sentence into tokens. Apart from that, lemmatization, part-of-speech tagging, parsing and mention detection were already conducted. Each mention was also assigned a unique character identifier from a knowledge base of 400 different characters. Those identifiers serve as labels for training our system and as gold keys for testing it.

[1] https://competitions.codalab.org/competitions/17310

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 1–5, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

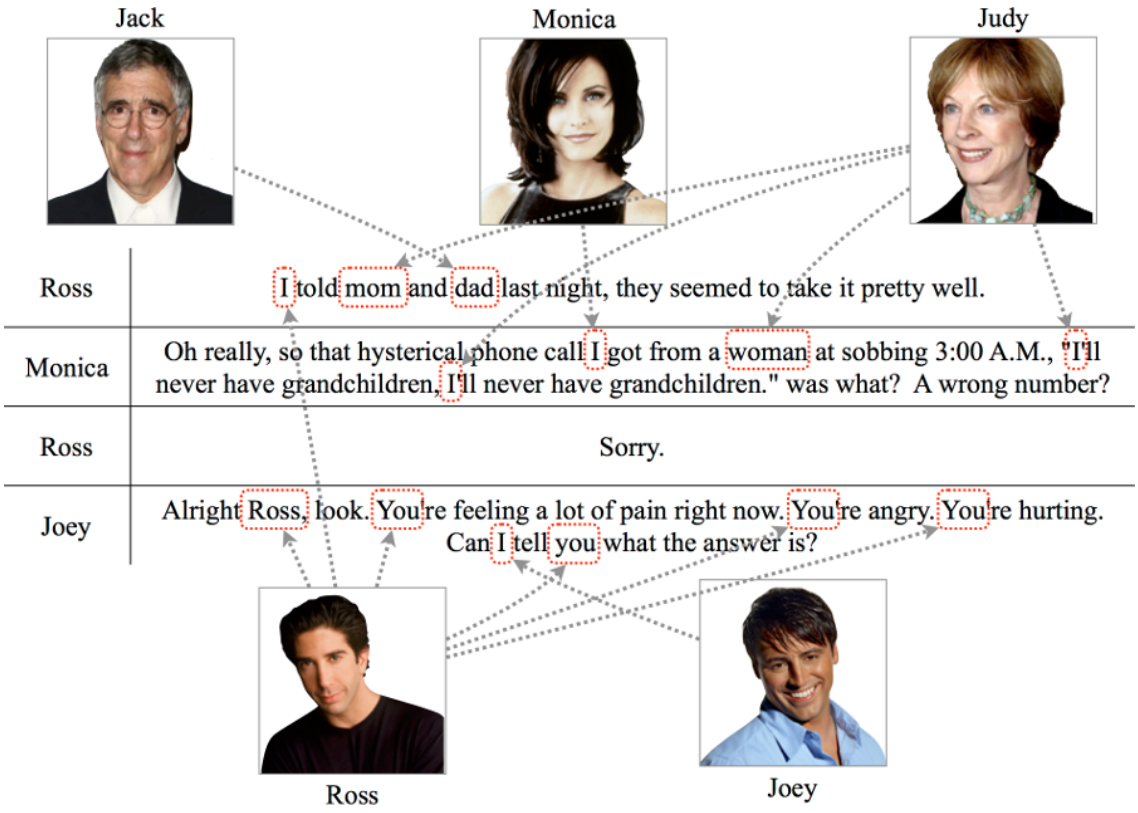


Figure 1: Example of a conversation with marked mentions that are mapped to the character they refer to. Taken from `https://competitions.codalab.org/competitions/17310`

There are 13,346 mentions in total and the distribution of their respective part-of-speech tags is shown in Table 1. Since most of the mentions are expressed as personal or possessive pronouns, we have identified all unique pronouns and manually arranged them into lists based on grammatical person and number. Those lists are useful for feature extraction and entity linking rules.

Table 1: Distribution of part-of-speech tags.

POS tag	Occurrences
PRP	9,112 (68%)
NNP	1,877 (14%)
PRP$	1,313 (10%)
NN	940 (7%)
Others	104 (1%)

We have further modified the list of mentions in order to include gender information (male, female or unknown) where possible. Each mention was processed according to the following rules.

1. If the mention is a third-person pronoun, the gender of the mentions is the same as the gender of the pronoun.
2. If the mention is a first-person pronoun, the gender of the mention corresponds to the gender of the speaker which was previously manually annotated.
3. If the mention is a noun or a noun phrase, its gender is extracted from the annotated number and gender data created in Bergsma and Lin (2006).

4. System Description

Our system is composed of two parts: (1) a mention-pair coreference resolution classifier that generates coreference chains and (2) entity linking rules that map each chain to one character.

4.1. Coreference Resolution

We decided to use a simple mention-pair model for tackling coreference resolution problem. The main idea behind this approach is to train a binary classification model for detecting whether or not a pair of mentions is coreferent.

First we have to generate mention pairs for training the classifier. One approach is to generate all possible pairs, which would result in a huge number of non-coreferent pairs polluting the training data. We chose the closest antecedent approach which, given a mention m_j, searches for the first preceding coreferent mention m_i and generates a positive pair (m_i, m_j) for training. For each mention m_k, where $i < k < j$, a set of negative pairs (m_k, m_j) is added to the training set. To address the previously described problem, search for the first antecedent stops at the beginning of the current scene. For example, if m_j is the last 'you' in Figure 1, then m_i would be the word 'you' from "You're hurting." and we would add one positive pair (you, you) and one negative (I, you) to the training set. This

method generated 58,330 pairs, out of which 11,436 were positive and 46,894 negative.

The next step is the extraction of features from each pair of mentions. Bengtson and Roth (2008) emphasize the importance of choosing the proper features for coreference resolution and propose a handful of generic features which we adapted for our problem. All features we use are of Boolean type and Table 2 provides their overview. Labels d_{sent} and d_{men} represent distance between a pair of mentions in sentences and mentions respectively. It is worth noting that both distances are encoded as multiple features and we chose to use $s_n = 3$ and $m_n = 4$ because adding more distance features did not improve the performance on the test set.

Table 2: Features by category.

Type	Description
Structural	$d_{sent} == s_i, \quad s_i \in \{0, \ldots, s_n\}$ $d_{sent} > s_n$ $d_{men} == m_i, \quad m_i \in \{1, \ldots, m_n\}$ $d_{men} > m_n$
Morphological	gender matching person of a pronoun
Lexical	part-of-speech matching lemma matching speaker matching

We decided to use the random forest classifier that is implemented in the *scikit-learn*[2] library and feed it with the aforementioned features extracted from each mention pair. Trained classifier is then used for generating coreference chains on the test data. A single coreference chain contains mentions that are coreferent to one another and is built in the following way.

1. Each mention m_j is inserted into the chain containing m_i which is the closest preceding coreferent mention based of the classifier output.
2. If there is no preceding coreferent mention in the current scene, m_j is inserted into a new chain.

4.2. Entity Linking

Once coreference chains are built, the next step is to map each chain to the corresponding character in our knowledge base. To accomplish that, each mention in a chain votes for one character in the following manner.

1. If the mention is expressed as a first-person pronoun or a possessive pronoun, it votes for the speaker.
2. If the mention is expressed as a noun phrase and there exists an entry in the knowledge base starting with that noun phrase, it votes for that character entry. If there are multiple acceptable entries, the first one is chosen. For example, a noun phrase "Chandler" votes for the entry named "Chandler Bing".

[2]http://scikit-learn.org/stable/index.html

After the voting process is done, each chain is mapped to the character that has received the most votes in that chain. If a chain has no votes, it is mapped to the *unknown* group.

5. Evaluation and Results

For all of the conducted experiments, we used the provided test set that contains transcripts from all 24 episodes with 2,429 mentions. Reported scores are measured with an official evaluation script that compares gold keys with our outputs.

To establish a baseline we computed the most likely estimates from the training data. We counted how many times each speaker referenced each character with each word. To obtain the baseline output on the test set, first we find a mention that needs to be mapped to a character. Then we look up what characters did the speaker refer to with that word and choose the most likely one.

Hyperparameter optimization, for our random forest classifier, was done with a 5-fold cross-validation on the training data. Optimal number of decision trees proved to be 10, each with maximum depth of 15. Furthermore, for evaluating the trained coreference resolution system, we computed some of the relevant metrics using the official CoNLL scorer[3] and the results are shown in Table 3. Importance of each reported metric and how they are calculated is explained by Cai and Strube (2010).

Table 3: Coreference resolution metrics.

Metric	Precision	Recall	F1
MUC	92.96	81.73	86.98
B^3	85.36	59.57	70.17
$CEAF_e$	40.92	72.42	52.29
$BLANC$	82.51	72.38	75.58

Table 4 shows the results of a system-wide evaluation split into two groups. First group, main entities, considers the 6 main characters as individual classes and all other characters as a single class. The second group, all entities, considers all characters that appear in both training and test sets as individual classes. Two metrics are computed for both groups: label accuracy and macro-average F1 score. First four entries are taken from the official competition site[4] and show the competitors' results. Top competitor in the main entities group seems to have specialized in identifying only the main characters and shows poor performance when considering all characters. Already our baseline shows competitive results, particularly in the main entities group. Probable reason behind that is a small number of main characters and a low variety of words used to express each mention. As we showed earlier most of them are expressed as pronouns. Finally, our work surpassed the results achieved by the baseline and offers the best average F1 score on all entities.

[3]http://conll.cemantix.org/2012/software.html

[4]https://github.com/emorynlp/semeval-2018-task4

Table 4: Character identification results on the test set.

	Main entities		All entities	
Model	**Label Accuracy**	**Average F1**	**Label Accuracy**	**Average F1**
Cheoneum	**85.10**	**86.00**	69.49	16.98
AMORE UPF	77.23	79.36	**74.72**	41.05
Kampfpudding	73.36	73.51	59.45	37.37
Zuma	46.07	43.15	25.81	14.42
Baseline	73.03	73.20	61.55	37.05
This work	78.75	78.80	66.15	**44.05**

5.1. Ablation study

To recognize the value of each feature in our model, we conducted an ablation study. We removed one feature group at a time and re-trained the model. Results of testing on the test set are shown in Table 5 with reported average F1 scores in all entities group. The highest performance loss was observed without features that carry lexical information, such as speaker matching. Without them, our system is unable to correctly infer which mentions expressed by first-person pronouns are coreferent.

Table 5: Ablation study, separately removing feature groups

Features	**Avg F1**
Full model	44.05
w/o sentence distances	-3.39
w/o mention distances	-1.75
w/o gender information	-2.25
w/o pronoun person information	-2.86
w/o lexical information	-5.78

6. Conclusion

In summary, we successfully combined a supervised mention-pair model with entity linking rules to tackle the character identification task. Compared to other competitors, we believe our system offers the best all-around performance as it achieves the best score when considering all entities. Due to the differences between the test sets that were used, we could not directly compare our system to the state-of-the-art model by Chen et al. (2017) that is based on more sophisticated deep learning approaches.

Inspection of chains that our system mapped to the *unknown* group reveals that they contain common nouns and pronouns such as: he, she and you. Unfortunately, there are no simple heuristics for inferring which character they refer to.

References

Eric Bengtson and Dan Roth. 2008. Understanding the value of features for coreference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 294–303. Association for Computational Linguistics.

Shane Bergsma and Dekang Lin. 2006. Bootstrapping path-based pronoun resolution. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 33–40. Association for Computational Linguistics.

Jie Cai and Michael Strube. 2010. Evaluation metrics for end-to-end coreference resolution systems. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 28–36. Association for Computational Linguistics.

Yu-Hsin Chen and Jinho D Choi. 2016. Character identification on multiparty conversation: Identifying mentions of characters in tv shows. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 90–100.

Henry Y Chen, Ethan Zhou, and Jinho D Choi. 2017. Robust coreference resolution and entity linking on dialogues: Character identification on tv show transcripts. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 216–225.

Kevin Clark and Christopher D Manning. 2015. Entity-centric coreference resolution with model stacking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1405–1415.

Kevin Clark and Christopher D Manning. 2016. Deep reinforcement learning for mention-ranking coreference models. *arXiv preprint arXiv:1609.08667*.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the fifteenth conference on computational natural language learning: Shared task*, pages 28–34. Association for Computational Linguistics.

Wee Meng Soon, Hwee Tou Ng, and Daniel Chung Yong Lim. 2001. A machine learning approach to coreference resolution of noun phrases. *Computational linguistics*, 27(4):521–544.

Sam Wiseman, Alexander M Rush, and Stuart M Shieber.

2016. Learning global features for coreference resolution. *arXiv preprint arXiv:1604.03035*.

Extraction of Drug-Drug Interactions

Tomislav Aščić, Darin Dašić, Pero Skoko

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{tomislav.ascic, darin.dasic, pero.skoko}@fer.hr

Abstract

DDI Extraction is a challenging problem as it requires specific knowledge of chemistry, pharmaceutics and medicine. Semeval 2013 Drug-drug interaction (DDI) extraction task contains of two subtasks. First one is named entity recognition (NER) and second one is extraction drug-drug interactions. Given gold standard data for both tasks, we have created one system for each task and presented our results in this paper. Tasks are independent of each other, while final goal is to create system dependent on both tasks.

1. Introduction

Researches in chemistry and pharmaceutics deliver new drugs almost every day so the drug database increases rapidly. Problem is that they are researched on small groups of people so researchers usually do not recognize all side effects of other drugs. This change of drug-effect could be dangerous and could even cause death so it should be well tested. Definition of drug-drug interaction (DDI) is decreasing, increasing or totally change in behavior of substances in drug when interacting one with some other drug. Reducing costs of drug research, healthcare and improving safety of patients are followed by well designed DDI system (Isabel Segura-Bedmar, 2013). Our DDI Extraction task, just like DDIExtraction2013, contained two subtasks, NER and DDI Extraction. Both subtasks had to be developed on the same dataset, composed from MedLine and DrugBank. Those databases are both big and contain thousands of entities but we used a part that was already preselected for DDIExtraction2013 competition. NER is important part for DDI because it locates and classifies words into their predefined categories. It is frequently used in all kinds of Natural Language Processing (NLP) tasks. Most common start is tokenization. It splits sentences into tokens, usually words, and then categorizes them so that further processing would work. Good tokenization and classification are very important so that model could be well trained. DDI Extraction decides if two drugs interact with each other. Just recognizing interaction is not enough, it has to declare if that interaction is dangerous for humans or not. Of course, those results should not be accepted without human recension and laboratory control but it could eliminate big number of researches and maybe potentiate research on interactions of some drugs that were not recognized before. DDI Extraction2013 competition had more aims. One was to develop new state-of-the art models for pharmacological domain and make significant improvement in that area. Other aim is to provide common framework for evaluation of systems for those tasks (Isabel Segura-Bedmar, 2013). This was second organized event of this type and it had brought good results. The best results got team FBK-irst and in best runs it was 80% for detecting DDI on all texts. Their results were much better when using DrugBank dataset only. We did not want to separate database so all our models and tests were on database that combined both DrugBank and MedLine datasets.

2. Related Works

There were many teams competing on DDIExtraction2013. Some of them, like previously mentioned FBK-irst, achieved very good results and made improvement on state-of-the art. Most of the teams used Support vector machine (SVM) models. SVM showed best results on DDIExtraction 2011 which is understandable but on the DDIExtraction2013 the best results got those who brought some new ideas and approaches to tasks.

FBK-irst team used SVM with a lot of discarding. They discarded less informative sentences and also less informative instances. Discarding less informative sentences was useful to minimize false negative examples. Discarding instances was done by considering false negative, while computing results. This was done on both datasets, training and test. That way they improved their results a lot.

Second best work was WBI-DDI team. Their F1 score was 78.6%. They also used SVM but with a little bit different tools than FBK-irst team. Their approach was to do the classification in two steps. First, they were only detecting interaction of two drugs and did not consider any classification. Then they reclassified result of first classification in four classes, as asked in task (Philippe Thomas and Leser, 2013).

3. Dataset

Dataset consists of two parts, one is from DrugBank and the other is from Medline. It is separated in xml structured files. Each file has tagged sentences and annotated sentences, entities and pairs. It has 6976 sentences in 714 files. Each file can and doesn't have to contain interactions of drugs and entities but most of sentences have more entities. That's why it has 14765 entities, but only 2972 different ones. One entity appears almost five times in dataset. DrugBank dataset has 5675 sentences and MedLine has 1301. 3789 sentences have declared some interactions between drugs. Our dataset is the same as the one on DDIExtraction2013 with difference that we couldn't get test data. That's why we splited our dataset in ratio 8:2 on files base. Out of 10 files 8 were used for training set and 2 were used for tests.

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 6–9, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

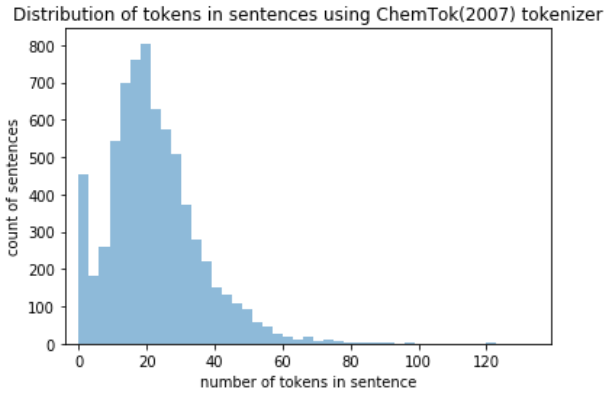


Figure 1: Model for *Distribution of tokens*

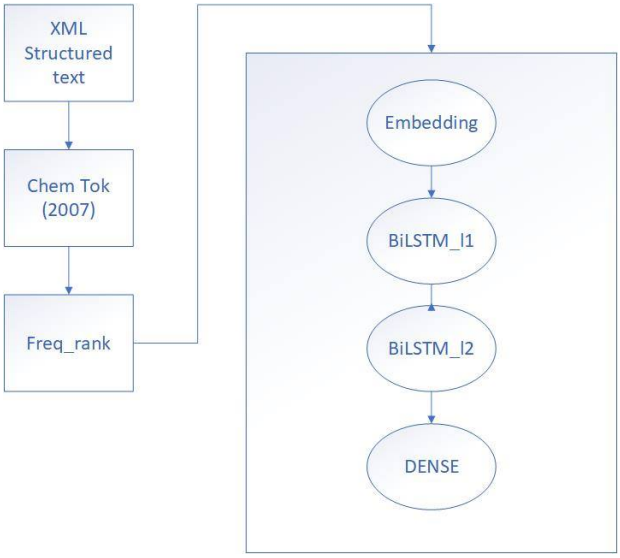


Figure 2: Model for *Named entity recognition*

Further, training set was split on the same way, 8 files were used for training and 2 files were used for validation.

4. First Task

For the drug name extraction our system consists of three steps. First we need to prepare data for our machine learning model. Next we apply training ready data to our bidirectional RNN. And at last our tagged tokens need to be transformed into file as instructed for evaluation scripts.

4.1. Data Transformation

Since our method uses machine learning we need to transform text to machine-friendly data. To match data for our model, each sentence is going to be tokenized. For tokenization we used whitespace tokenizer and ChemTok from OSCAR(Open Source Chemistry Analysis Routines)[1], not to be confused with ChemTok (A. Akkasi and Dimililer, 2016). ChemTok has given better results since it is built for tokenization of biomedical and chemical texts. After we get token for each sentence, we build a dictionary and rank each token by its frequency in our document. After doing so, our tokens are ready to be fed into embedding layer. We used embedding layer to transform each token into a dense vector representation of a word. Tokenization is very important part for building our system. As described in ChemTok 2017 [abassi] their ChemTok tokenizer has yielded 88.65% while whitespace tokenizer produced 77.89 F score for DrugBank data while for MedLine difference was even greater 64.88 vs 51.51 F score for 9.1 task. We experimented with different length vectors and found that 256 element vector worked the best.

For this NER task we had to label each token with correct class. Our task was to find drug, drug_n, brand and group enitites. Because some entites are spanning through multiple tokens, we used BIO tagging scheme. To each label we add suffix "-B" if this is the begining of an entity or "-I" if this token is in the middle, or the end of the entity. For example entity "contraceptive drugs" is labeled as group in gold standard data and our system would tokenize it and label into ("contraceptive", "group-B"), ("drugs", "group-I").

While entity "insulin" is tokenized and labeled as ("insulin","drug-B"). Tokens that do not fall into any of the given categories are tagged with label "O". Our ML model takes text as input with fixed number of tokens. So we decided to find the text with the maximum number of tokens in our training data, and use its length as fixed size. Sentences with less tokens are padded with fill tokens and tagged with "O" label.

4.2. Machine Learning

For our ML model we decided to use bidirectional recurrent neural networks. We opted for LSTMs as cells. Then we had to find the best hyperparametars for our model. To do so we ran tests with different paramaters. Using validation data we found that the model with 2 BiLSTM layers, each with 16 units, using Geoffrey Hinton's RMSprop produced the most consistent results. We implemented this model using Keras framework. For loss function we used categorical crossentropy and at the output we used softmax layer.

4.3. Evaluation

Official rules proposed evaluation based on 4 different scorring: Strict evaluation, exact boundary matching, partial boundary matching and type matching. For the task 9.1 we failed to follow official evaluating steps. While we created all neccessary steps for creating output in format needed for the script, running the evaluation scripts resulted in errors. Unfortunatly, we failed to correct them.

While finding the best hyperparametars we used validation data to decide when to stop training our model and which hyperparametars provided best results while being consistant. We based our decisions according to validation accuracy which was about 0.9970. To remind the reader again, this accuracy is far from good since most of the classes in output are the same class, "O" since most tokens in text are not any of the drugs, groups or brands.

[1] https://sourceforge.net/projects/oscar3-chem/

Table 1: Results for task 1

	precision	recall	f1-score	support
Overall	1.00	0.99	0.99	26314
drug_n-B	0.82	0.78	0.80	103
drug_n-I	0.73	0.42	0.54	26
group-B	0.89	0.92	0.90	621
group-I	0.88	0.92	0.90	435
brand-B	0.97	0.86	0.91	291
brand-I	0.86	0.46	0.60	13
drug-B	0.92	0.97	0.94	1846
drug-I	0.77	0.93	0.85	120

So to give an aproximate idea of the performance of our model we present in table 1 scores of our LSTM network, and how it was able to classify given tokens to the wanted tag.

The achieved results are on 1

In this task we used approaches defined in (A. Akkasi and Dimililer, 2016), (J. Bjorne and Salakoski, 2013) and (Vijaya, 2017).

5. Second Task

In this task we used approaches from (Sun Kim and Wilbur, 2015) and (Victor Suarez-Paniagua and Martinez, 2016). First paper uses SVM with linear kernel and rich set of hand designed features. Second paper uses the convolutional neural network which learns all features itself.

5.1. SVM with Linear Kernel and Rich Set of Features

Aproach using SVM with linear kernel used variety of features. First type of features were word level features which used n-grams of size up to three to represent words. To represent patterns involving distant words, word pair features were used which were composed of unigram pairs of words which appeared some minimum number of times and which had some maximum value p calculated using hypergeometric distribution. Those word pairs appear in sentences which are labeled as positive or negative, depending on whether some interaction between some entities is present in those sentence. Therefore, in training dataset, it is possible to mark all appearances of some word pair as positive or negative and calculate probability of such occurrence of that word pair in training dataset using hypergeometric distribution. High p value indicates that distribution of occurrences of word pair across positive and negative sentences is close to uniform and that such word pair can be discarded, but the low probability indicates that occurrences of those words are not random and that they provide some information about interactions between entities. The third type of features were encoded from shortest paths between words in dependency graph. To enrich them, features extracted from parse tree were used. The last type were features representing noun phrases.

•To generate such features, including the dependecy parse trees and noun phrases, we used source code provided by authors of (Sun Kim and Wilbur, 2015) which can be found here: `http://www.ncbi.nlm.nih.gov/IRET/DDI`

The input data was preprocessed and sent to linear SVM with the following cost function:

$$C = \frac{1}{2}\lambda|w|^2 + \frac{1}{T}\sum_{i=1}^{T} h(y_i(\theta + w \cdot X_i))$$

First expression is regularization term and second is the sum of error across all examples where h is defined as:

$$h(z) = \begin{cases} -4z, & \text{if} z \leq -1 \\ (1-z)^2, & \text{if} -1 < z < 1 \\ 0, & \text{if} 1 \leq z \end{cases}$$

That is the classification variant of Huber loss function. The $y_i(\theta + w \cdot X_i)$ is the product of correct classification (-1 or 1) and output of SVM so second and third row of upper formula are standard hinge loss used for SVM training. The value of such loss differs from hinge loss when the product $y_i(\theta + w \cdot X_i)$ is less than -1. Usage of modified Huber loss function is motivated by the fact that authors in (Sun Kim and Wilbur, 2015) noticed that such function consistently achieves better results on biomedical classification problems than standard hinge loss function.

The results were:

All	DEC	MEC	EFF	ADV	INT
0.670	0.775	0.693	0.662	0.725	0.483

5.2. Convolutional Neural Network

Second approach is based on convolutional neural network. The input sentence was preprocessed where drug mentions were replaced with generic names like *drug1 drug2*. Then the input matrix was generated which was suitable for convolution which had the same size for all input examples and that size was defined as the size of the largest sentence. For smaller sentence, input matrix was padded with zeros.

Words were represented with vector with random initialization. That means that for each word its unique vector representation was randomly generated without any training. Besides the word embedding, the position of word was also embedded. Position was the distance of both drugs whose interaction was being asked for. By concatenating those embedding we got the input vector.

On that input matrix two dimensional convolution was applied which extracted features which was than pooled and classified using softmax layer with droput. As a loss, standard negative log expectation was used:

$$J(\theta) = \sum_{i=1}^{T} logp(y_i|\mathbf{x}_i, \theta)$$

There were 200 filters with all having the same size 6 in convlutional layer. The softmax layer had a dropout of 50%, and the network was trained using backpropagation with standard gradient descent.

It achieved the following F1 scores on whole dataset:

All	MEC	EFF	ADV	INT
0.604	0.631	0.584	0.663	0.429

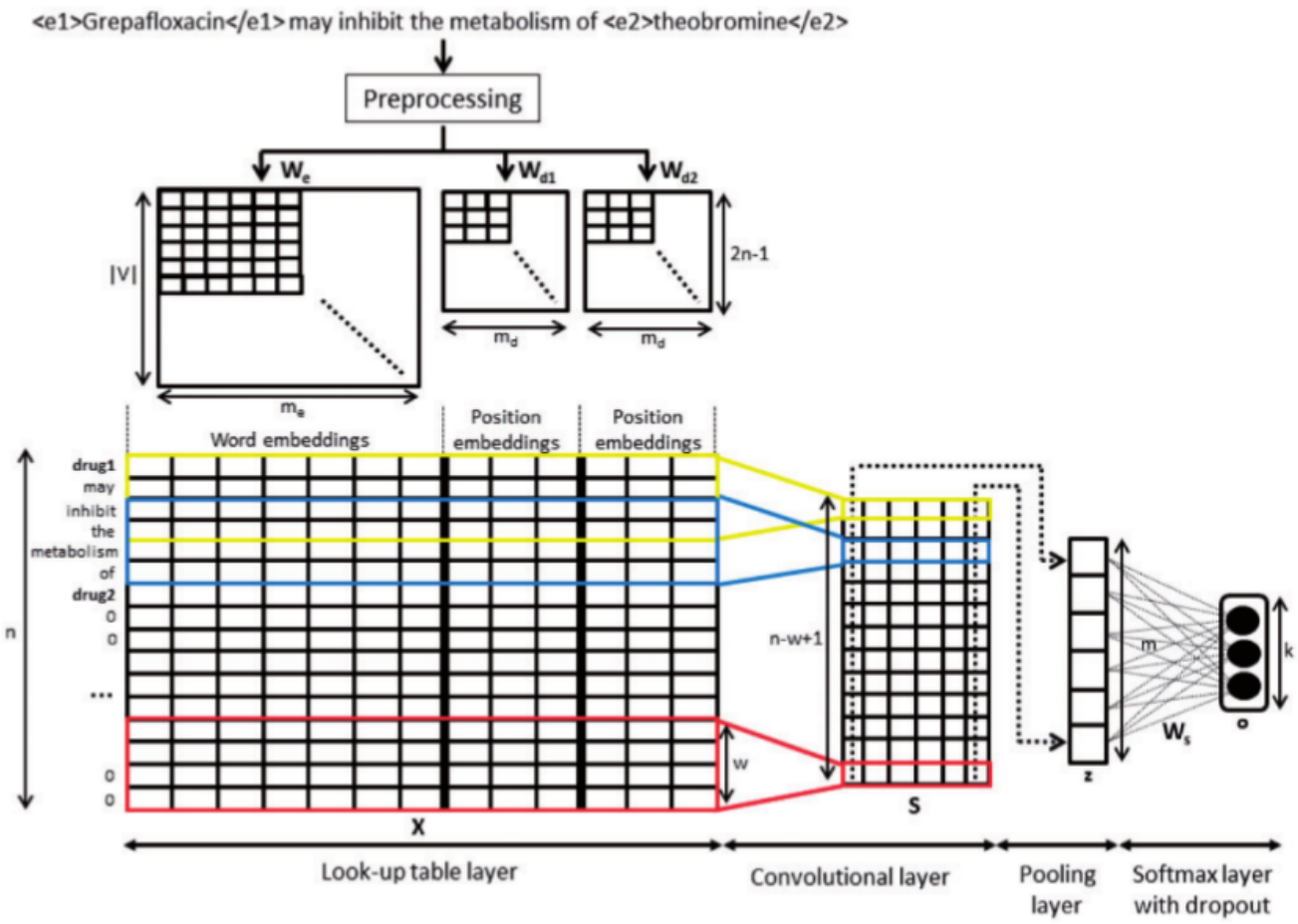


Figure 3: Convolutional neural network; taken from (Victor Suarez-Paniagua and Martinez, 2016)

6. Conclusion

Our task had two parts, drug name extraction and drug-drug interaction. We were given annotated data and created two models , one for each task. Model for the first task did not provide results which are comparable to other systems. Which means if we created an end-to-end system which takes outputs from the first task as the inputs for the second task, it would perform poorly since the first model is weak chain. For the task 9.1 in future we will consider models which use BiLSTMs together with Conditional Random Fields. Since other works reported state of the art results for the said systems. Another aspect to consider is text representation in which we could use sets of features which most referenced works used in their models.

In second task we compared the performance of a linear SVM with rich set of hand designed features with convolutional neural network. SVM had a slightly better performance but convolutional network was not far with its results. This proves that automatically learned features with convolutional network can have performance comparable to carefully hand designed features. This is important because convolutional networks model was simple to design. In future we would like to test multiple variants of RNNs which might be more suitable for natural language processing task.

References

E. Varoglu A. Akkasi and N. Dimililer. 2016. Chemtok: A new rule based tokenizer for chemical named entity recognition.

Maria Herrero-Zazo Isabel Segura-Bedmar, Paloma Martinez. 2013. Semeval-2013 task 9 : Extraction of drug-drug interactions from biomedical texts (ddiextraction 2013). *Second Joint Conference on Lexical and Computational Semantics (*SEM)*, 2:pages 341–35.

S. Kaewphan J. Bjorne and T. Salakoski. 2013. Uturku: drug named entity recognition and drug-drug interaction extraction using svm classification and domain knowledge.

Tim Rocktaschel Philippe Thomas, Mariana and Ulf Leser. 2013. Wbi-ddi: Drug-drug interaction extraction using majority voting. *Second Joint Conference on Lexical and Computational Semantics (*SEM)*, 2:pages 628–35.

Lana Yeganova Sun Kim, Haibin Liu and W. John Wilbur. 2015. Extracting drug-drug interactions from literature using a rich feature-based linear kernel approach.

Isabel Segura-Bedmar Victor Suarez-Paniagua and Paloma Martinez. 2016. Exploring convolutional neural networks fordrug–drug interaction extraction.

S. Vijaya. 2017. Biomedical named entity recognition - a swift review.

Partitioning the Task of Drug-Drug Interaction into Meaningful Subtasks

Jure Baban, Viktor Golem

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{jure.baban, viktor.golem}@fer.hr

Abstract

This document tackles the SemEval 2013 task of extracting pairwise pharmaceutical drug interaction information on a sentence level from biomedical texts. First, all the drug entities are identified and extracted, after which they are to be labeled in one of the categories. Then, their interaction type, if exists, is to be extracted from the sentence. We try solving this problem in a number of ways, by experimenting with different subtask partitions, along with different models, such as an ensemble of shallow classifiers and deep neural networks, and a sequence labeling model. Our goal is to determine whether it pays of to partition one multi-label classification task into a two-stage binary - multi-label approach, for both of our main tasks. We present our results for each subtask and model variation.

1. Introduction

This paper focuses on the DDIExtraction task presented in the SemEval 2013 competition. DDI (drug-drug interaction) is broadly described as a change of effect of a single drug when in the presence of another drug[1]. Detection of DDIs is beneficial both for consumers, as drug interactions can present serious health risks if unmanaged; and for the pharmaceutical industry for their easier extraction and indexing from biomedical texts. Main goal of this paper is implementing a model which can recognize drug names from biomedical texts and correctly assign them one of the labels, and then extract drug-drug interactions which occur between found drug entities. Note that drug-drug interactions are always defined only on pairs of drugs (pharmaceutical substances). Furthermore, this paper proposes several different subtask partitions of given tasks, along with different model configurations. For example, the first (drug entity extraction) task can be formulated both as a single multiclass-classification problem, or a combination of a binary classification problem (whether a given entity is a drug or not) and, if an entity is classified as a drug, which of the four different classes it belongs to. These differing arrangements make for varying models giving varying results, which are described further. Similar partitioning is described for the second, drug interaction extraction task.

Another consideration during task partitioning and model creation was whether to use sequence labeling or not. Assuming that the input is a sequence of instances (in this case, word tokens), and model output is a sequence of labels, traditional algorithms assume that each token is independent of its surroundings, while sequence labeling models assume that token labels are dependent on the sequence labels. Moreover, this concept is implemented using a deep recurrent neural network, which were not often considered at the time this task was first published. In sharp contrast to deep models, this paper also explores the possibility of solving the subtasks using traditional classifiers with several features.

[1]Extraction of Drug-Drug Interactions from BioMedical Texts https://www.cs.york.ac.uk/semeval-2013/task9/

2. Related Works

Of all the papers of this topic that we surveyed, many of them are based on traditional machine learning algorithms, but some of them have a greater emphasis on natural language processing (NLP) methods, such as named entity recognition (NER) than others. Bedmar et al. (2013) propose the main subtask division (drug classification and interaction extraction). They note the advantages of non-linear kernel-based methods over linear SVMs for extraction of DDIs. Furthermore, they note difficulties in recognizing active substances not approved for human use ('DRUG-N' category, discussed further in the text). Bjorne et al. (2013) applied the Turku Event Extraction System 2.1 (TEES) for detection of drug names and drug-drug interactions. The system works by processing text in a pipeline of components (named entity recognition (NER), parsing, event extraction, etc.) but note the lower expected performance in a real-world scenario. They also show that adding additional external data from biomedical text sources (DrugBank, Open Data Drug, Drug Target database) considerably improves model performance, however the problem of classifying new, unseen pharmaceutical entities is still present. Bui et al. (2014) propose a feature-based three step approach for this task. After representing sentence with vectors, they map each candidate DDI pair from that dataset into a suitable syntactic structure. Then, they generate feature vectors from those DDI pairs and classify using SVM. However, we did not come across much work that utilizes deep learning models, or ensemble methods of any kind.

3. Dataset

DDIExtraction 2013 task is accompanied by the DDI Corpus[2], a semantically annotated corpus of documents describing drug-drug interactions collected from DrugBank database and MedLine abstracts. It is intended for training of information extraction tasks of this domain, and is manually annotated with pharmaceutical substances (drug entities) and interactions between them (where they exist) for

[2]https://www.cs.york.ac.uk/semeval-2013/task9/data/uploads/the-corpus-ddi.pdf

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 10–14, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

each sentence from documents or abstracts. All of the annotation work was carried out by the Advanced Databases Group (Labda), from Universidad Carlos III de Madrid, Spain.

Data is publicly available and is provided in XML format. All of the data is annotated, meaning that no test data is provided by default. There are in total 572 DrugBank documents and 142 MedLine abstracts in the corpus, with an average of 2.3 sentences per document and 1.4 sentences per abstract. Majority of those sentences have 'entity' XML tags attributed to them, indicating which drug entities are present in each of them. Those entities can span multiple words, and are given their label (type), character offsets (indicating exactly where they appear in a given sentence). If an entity is classified as a drug, it belongs to one of four categories (drug types): 'DRUG' (generic drug names), 'BRAND' (branded drug names), 'GROUP' (drug group names), 'DRUG-N' (active substances not approved for human use). All of the tags, such as entities, sentences and documents also have their unique identification as an attribute.

Along with 'entity' XML tags, sentences can have multiple 'ddi' XML tags indicating an existence of a drug-drug interaction between a pair of entities. Entities which this refers to are defined by their identification attributes in a 'ddi' tag, along with a label defining which category this interaction belongs to. Possible categories (interaction type) for DDIs are: 'ADVICE', 'EFFECT', 'MECHANISM', 'INT'. Note that all interactions are annotated at the sentence level, so DDIs spanning multiple sentences are ignored. A detailed explanation of all categories is available at the task description site. Both DrugBank and MedLine data is formatted the same but are characterized by different writing and language styles of sentences, with the latter being much more technically written, and the former comparable to the language style used in package inserts, as noted by Bedmar et al.

4. Models

As mentioned before, the main assignment of DDIExtraction is split into two distinct tasks. Given a sentence, a model is supposed to (1) recognize and classify pharmaceutical entities (drugs) from each of the sentences if there are any present; and (2) identify and correctly classify interaction for pairs of drugs extracted from a sentence. We treat and evaluate these two tasks with separate models and using different data. However, we discuss the notion of combining all of them into a single solution. Input to that model would be a sentence from a biomedical text, and the output would be a formatted document containing all the found entities, their interactions and positions in a text. It is certainly achievable by treating the output of task (1) as input examples, along with the sentences for our task (2) model(s).

4.1. Task 1

Task 1 is essentially a named entity recognition task, specialised for the domain of biomedical texts and pharmaceutical entities. We approached the problem both by treating the task 1 as a single task, and by partitioning it into smaller subtasks. Our intuition was that handling the drug extraction and labeling problem would perform better as two separate tasks, since training a single binary classifier just to label whether a sentence part is a drug or not (from now on refered to as task 1a) would greatly offload the workload of the second-stage classifier, dealing with labeling the found drugs (task 1b). The fact that there is an average of only 2.3 and 1.4 drugs per sentence on DrugBank and MedLine datasets, respectively, supports this theory, since a properly trained first-stage classifier would eliminate most of the sentences as not being a part of a drug entity. However, we were unsure about whether this is the case in a real-world scenario, so we also experimented with testing a single multi-class classifier for this task.

Our method first involved reading and parsing the provided XML datasets described in Section 3. We treat sentences as a collection of ordered word tokens. Traditional shallow models use tf-idf encodings for word and sentence vectorization, while deep models use pre-trained GloVe embeddings[3]. As such, a new dataset is created where each word token of a sentence is a training example. In the context of a unified task 1, target labels are 'NO-DRUG' if a word token is not a part of a named drug entity, or one of the other labels ('DRUG', 'BRAND', 'GROUP', 'DRUG-N') if a word token is a part of the named entity. This model is also given an additional feature in the form of the whole vectorized sentence to which individual word tokens belong, that functions as the context of a token. We experimented both with a single linear SVM classifier and an ensemble of multiple classifiers (LSTM, feed-forward neural network, random forest, logistic regression and linear SVM classifiers) for this task.

On the other hand, if we partition task 1 into task 1a and task 1b, for each of them we propose different models and different datasets. Task 1a is also based on a word-token dataset as is the case for task 1, with the vectorized sentence as the context feature, and POS (Part of speech) tags as an additional model feature. Target labels are either 'NO-DRUG', or 'DRUG'. This stage was also experimented with using an SVM classifier. As an assumption was made that word context, its position and surrounding in a sentence is an important determiner of whether that word token belongs to a drug named entity or not (surrounding here is encoded by the vectorized sentence feature), we also proposed a different approach. It is based on sequence labeling, which directly takes into account the labels of surrounding tokens for the purpose of classifying the current token. Sequence labeling is implemented in the context of a LSTM (Long-short term memory) recurrent deep neural network.

After this binary classification stage, tokens classified as 'DRUG' that appear in succession in a single sentence can be combined to form a multi-word drug entity. Exceptions to that rule are made, however, when that token sequence contains symbols such as ',', '.', '?', '!', since those tokens can also be mistakenly classified as a 'DRUG'. We now procede to task 1b. Training dataset here is similar to the

[3]GloVe: Global Vectors for Word Representation `https://nlp.stanford.edu/projects/glove/`

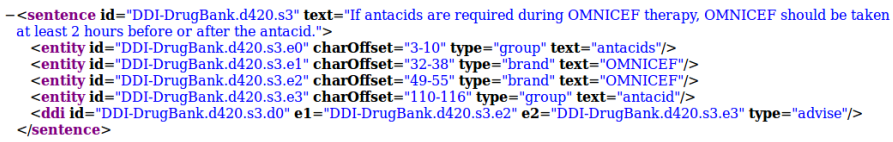


Figure 1: Example of a sentence in DrugBank dataset.

one described for task 1a, but instead of each word-tokens, extracted drug entities are now examples. They are also vectorized as described before, and are given both vectorized sentence context (each drug entity is assigned only the sentence it is derived from). Linear SVM and an ensemble of multiple classifiers (such as the one in task 1) are utilized and compared as classifiers in this subtask. Target labels here are 'DRUG', 'BRAND', 'GROUP', 'DRUG-N'.

4.2. Task 2

The goal of task 2 was determining what type of interaction, if any, does any given pair of drugs have. Firstly, it is important to state that the classification is considered correct if the resulting class matches the one mentioned in the given sentence (not if the type of interaction is really medically correct). This is important to have in mind especially for the 'NONE' label, which only represents the fact that an interaction between this pair of drugs has not been mentioned in the observed sentence. Since in the given XML files we only had the interactions of drugs which actually interact we looped throughout all sentences, took all pairs of mentioned drugs and assigned 'NONE' label for each pair which does not have any other label. After doing this, we observed that the label distribution is far from balanced. As expected, most pairs of drugs had 'NONE' label and most of other labels were in the 'EFFECT' and 'MECHANISM' class. We tried mitigating this issue by adjusting 'class weight' hyperparameters for all of our shallow models and we observed that we can not do better in terms of the F1 score, than putting this parameter to 'auto'. After that, it was a standard classification problem which we solved using voting combination of LSTM, feed-forward neural network, random forest, logistic regression and linear SVM classifiers.

Input to all models except for LSTM were tf-idf vector representations with unigrams of the sentence, first drug entity and second drug entity (so three concatenated vector representations). Only one tf-idf vectorizer was used for all of them and it was fitted on the sentences of the training part of the dataset so that each position in the all three vectors will correspond to the same word. For LSTM, we first concatenated sentence and pairs of drugs as strings and then used word embeddings. Our logic was that by doing this we will have some context of the sentence but since pairs of drugs in question would be at the end of the sentence they might have a greater role which is probably benefitial. The loss function for this approach was cross-entropy.

After doing this, we tried splitting this problem into two parts using in a similar manner to task one. First we performed binary classification where the goal was to determine if the drugs interact. Then we performed multiclass classification only for the interacting drugs to determine the type of interaction.

4.3. Model tuning

Although usually we would optimise hyperparametars for any given task, because of time constraints, we sometimes used the same hyperparameters for similar task. The hyperparameters were determined using cross-validation, with an 80-20 train-test split. In this paper we used five standalone models:

- Logistic regression with value of 'C' parameter optimized for any given task.
- SVM with linear kernel and value of 'C' parameter optimized for any given task.
- Random forrest with 30 estimators, 'min samples split' equal to 6, maximum number of features equal to 10%.
- Simple feed-forward neural network with just one hidden layer of size 100.
- LSTM - with both droput and recurrent dropout set to 0.2 and ADAM optimizer with learning rate of 0.01. Number of epochs was different in different tasks but it has never been set to a large number since the scores on the test set would get worse.

We also experimented with an ensemble of classifiers. Precisely, a simple voting classifier of the previously explained models. If there was more time we would try stacking as well since it usually works somewhat better but the difference is usually not that significant and voting classifer is much easier to implement (also, it does not require choosing and optimization of the metaclassifier).

5. Results

Initially, all the sentences for both datasets were split into train and test sets, in a 80:20 ratio.

5.1. Task 1

After running various combinations of described models and subtasks in previous chapter, the results are shown in Table 1. Table shows the subtask, model arrangement, accuracy and macro-averaged F1 scores. We ran task 1a both with linear SVM and voting ensemble classifiers and saw no real improvements with the ensemble, so we left it out from our evaluation. Also, we experimented with adding POS tags as features for our models, but saw no significant improvements. The scores of task 1 as a single task (task 1 in the table) and partitoned in two subtasks (task 1a, task 1b) are not directly comparable. Task 1b treats each entity drug name individually (it can consist of multiple word tokens), while the unified task 1 treats each word

token individually. Contrary to our intuition, it appears that the unified task 1 score is a better score in terms of the final performance of this model. That is because task 1a and task 1b were trained seperately, so we need to account the fact that task 1a errors (misclassifications) propagate to task 1b, so its scores would objectively be lower than the ones in the table. These conclusions should be taken with caution, because, as already stated, the scores are not directly comparable.

We experimented with sequence labeling for the binary classification task 1a, and despite tha high accuracy, we discovered that all the words were classified in the largest 'NO-DRUG' class. Modifying the individual class weights parameters mitigated this issue to some extent, but that model still performed worse than the SVM, as can be seen in Table 1.

Task	Model	Remarks	Acc.	F1
1a	linear SVM	no POS tags	0.899	**0.815**
1b	linear SVM	with POS tags	0.836	0.496
1b	voting	-	0.877	0.616
1	linear SVM	no POS tags	0.92	0.503
1	voting	FF, log.reg., SVM	**0.947**	0.572
1a	sequential	-	0.919	0.52

Table 1: Results (accuracy and F1) of various model combinations for task 1 (1 in table), task 1a (1a), and task 1b (1b).

5.2. Task 2

Table 2 shows results for task two. In accordance to our intuition , treating this as two subtasks improved the results. The final model for both subtasks and combined solution was a voting classifier, with just one difference: the LSTM has not yielded satisfactory results for combined solution so it was left out from voting classifier for this task. It is important to mention that LSTM was not crucial for other subtasks (without LSTM f1 score for task 2 of the voting solution was 0.6% lower) so the two-parts solution would be better without LSTM as well.

Task	Model	Remarks	Acc.	F1
2a	voting	LSTM, FF, log.reg., SVM, RF	0.755	0.676
2b	voting	LSTM, FF, log.reg., SVM, RF	**0.88**	**0.817**
2	voting	FF, LR, SVM, RF	0.758	0.445

Table 2: Results (accuracy and F1) of various model combinations for task 2 (2 in table), task 2a (2a), and task 2b (2b).

5.3. MedLine Dataset

Table 3 shows the unified task 1 model performance on two of the provided datasets. Our model was seperately trained and evaluated on both datasets. Because the MedLine dataset is more technically written and much smaller (1301 sentences, whereas DataBank has 5675), much more entities were not ever observed in the training process, its performance scores were expectedly lower.

Dataset	Acc.	F1
DrugBank	**0.947**	**0.57**
MedLine	0.92	0.368

Table 3: Results (accuracy and F1) of unified task 1 model on both datasets (using voting solution).

6. Conclusion and Future Work

Since there are millions of combinations of drugs in the world, knowing which drug entities interact and what is the nature of this potential interaction is quite challenging. Coping with this problem would be a bit easier if we had systems which would be able to return relevant information about interaction of the pairs of drugs. To make that possible it is necessary to be able to extract information from the raw text, such as which drugs are mentioned, which drug interactions are mentioned and what is the type of interaction. We also had to label drug entity either just as single drug, brand of drugs, drug group names, or substances not approved for human use.

From this perspective one could say that we had four tasks, namely extracting drug entities, classifying drug entities, extracting drug interactions and classifying interactions to several types. We explored a variety of subtask partitions for this main problem, and at the end the best results where achieved both by combining (task 1) and by separating the subtasks (task 2) in separate stages.

Dividing the multi-class problem into smaller subtasks sometimes makes sense but it is very important to have in mind that there should be a reason for doing so, not just from human perspective, but from perspective from machine learning algorithm, too. For example, if something is just a multi-class classification for the machine learning algorithm and if all the classes are equally balanced, dividing this problem into subtasks will not help even if there is some intuition behind this. This is why we think we get different conclusions on whether to partition the main task for both task 1 and task 2. Although there is intuition for partitioning from human perspective, from the point of machine learning algorithm, it appears that this can just be an unnecessary split of the one multi-class classification problem, as was the case for our task 1.

Future work could potentially benefit by incorporating ChemSpot[4] named entity recognition tool specialized for extracting mentions of chemicals in texts, similar to our problem. It utilizes a combined approach of employing a Conditional Random Field and a dictionary, pattern-based recognition and several methods for consolidating all annotations. It is easily to implement in Python through MolMiner library[5].

[4] https://www.informatik.hu-berlin.de/de/forschung/gebiete/wbi/resources/chemspot/chemspot

[5] https://gorgitko.github.io/molminer/py-modindex.html

References

Segura-Bedmar, Isabel and Martínez, Paloma and Zazo, María Herrero. Semeval-2013 task 9: Extraction of drug-drug interactions from biomedical texts (ddiextraction 2013) Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013).

Björne, Jari and Kaewphan, Suwisa and Salakoski, Tapio. UTurku: drug named entity recognition and drug-drug interaction extraction using SVM classification and domain knowledge Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013).

Bui, Quoc-Chinh and Sloot, Peter MA and Van Mulligen, Erik M and Kors, Jan A. A novel feature-based approach to extract drug–drug interactions from biomedical text Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh Oxford University Press, 2014.

Solving Community Question Answering Ranking Problem Using LightGBM

Borna Bejuk, Karlo Brajdić, David Emanuel Lukšić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{borna.bejuk,karlo.brajdic,david-emanuel.luksic}@fer.hr

Abstract

In this paper we describe our approach to solving the SemEval 2017 task 3 subtasks which are question-comment and question-question ranking problems important for the community question answering websites. We present both classification-based approach, using logistic regression, and ranking-based approach, using LightGBM framework. We used a variety of features ranging from simple cosine similarity between Tf-Idf vectors to more complex WordNet-based features. Our systems produce competitive results regarding the SemEval 2017 task 3 competition.

1. Introduction

Today, a great many people rely on community question answering (CQA) websites such as StackOverflow or Yahoo! Answers for satisfying their information needs. On the one hand, these services offer flexibility and freedom to users to ask any kind of question and receive a variety of answers. On the other hand, answers usually suffer from poor quality due to openness of the community and lack of moderation. Users are then left to manually search through the answers in hope of finding the information they desire. Therefore, the advantage of having a system that can automatically rank answers in a way that increases the probability of user finding the appropriate answer is obvious.

Over the years, the challenge has attracted a lot of attention from the research community. Consequently, it has been recognized that the community would benefit from having a common framework for comparing different approaches. For this reason, SemEval 2017 competition has included Task 3 with five subtasks, the main ones being Question-Comment and Question-Question similarity. In this paper, we describe our approaches to subtask A (Question-Comment similarity) and subtask B (Question-Question similarity). Dataset that is provided by the organizers was obtained from the Qatar Living web forum where people can ask different questions regarding their daily lives in Qatar. Organizers also prepared annotated data and evaluation method (MAP score) that is used to score the system.

Naturally, we formulated the task as a binary classification problem. We tackled the problem with two approaches. Firstly, with logistic regression and secondly with LightGBM (Ke et al., 2017), implementation of a gradient boosting decision tree (GBDT). LightGBM has proven to be a better fit for the task, justifying the praise that surrounds GBDT models. Features we used mostly follow the work of Šarić et al. (2012) and Šaina et al. (2017). Additionally, we experimented with doc2vec (Le and Mikolov, 2014) sentence vector embeddings.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of the dataset and subtasks. A description of our system and features is given in section 3. We present the results in section 4 and take a look at potential future work in section 5. Finally, the conclusion is given in section 6.

2. Subtasks and Data Description

The SemEval 2017 competition included five subtasks. Our paper deals with first two, namely A and B subtasks. We will briefly describe them.

2.1. Subtasks

Subtask A *Question-Comment Similarity.* Given a question Q and a comment list C consisting of ten comments in its question thread $(c_1, \ldots, c_{10})$, the goal is to rank these comments according to their relevance with respect to question Q. It is important to emphasize that this is a ranking problem and not a classification one. Organizers decided to choose this setting as it is closer to the application scenario. Accordingly, *Mean Average Precision* (MAP) was chosen as an evaluation measure. The aim of the system is to place all "Good" comments above the "PotentiallyUseful" and "Bad" comments. Latter two are not being distinguished at evaluation time, i.e., both are considered "Bad".

Subtask B *Question-Question Similarity.* In this subtask we are given question Q and a set of first ten related questions $(Q_1, \ldots, Q_{10})$ retrieved by a search engine. The comment threads of the related questions are also available. The objective is to rank related questions according to their similarity with respect to the original question Q. Here, we want the system to rank all "PerfectMatch" and "Relevant" questions above the "Irrelevant" questions, with first two being considered "Relevant". Same as in subtask A, MAP measure is used to evaluate the performance of the system.

2.2. Dataset

Dataset that we used was provided by the organizers of the SemEval 2017 competition and can be found on the competition's website.[1] All files, *train*, *dev* and *test* are in an xml format that is parsed to extract the information. The xml file is organized as a sequence of *original questions* that are each followed by 10 related annotated threads. Each thread

[1] http://alt.qcri.org/semeval2017/task3/index.php?id=data-and-tools

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 15–19, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Figure 1: Example dataset. From this example and many more like it, we can see that there is a lot of noise in the forum threads.

consists of a *related question* and ten comments posted as answers to the related question. Original question consists of a subject, a body and a unique question identifier (ORGQ_ID). Organizers included metadata about the related questions and the comments, but not about the original question in order to emulate a scenario where it is a new question, never asked before in the forum. Metadata available for the related question that we used in our system is:

1. RELQ_USERID: unique user identifier for the user that asked the related question
2. RELQ_CATEGORY: related question category, e.g., *Visas and permits*

Moreover, the only metadata about the comment that we exploited is:

- RELC_USERID: unique user identifier for the user that posted the comment

3. System Description

Because we have labeled data for each question-comment and question-question pair, we decided to frame this ranking problem as a supervised classification task. Simply put, classifier takes in a set of features that represent question-comment or question-question pair and produces an output that is a binary class with values *true* or *false* indicating whether the comment (or question) is relevant to the given question (true) or irrelevant (false). For the task of classification we decided to use simple logistic regression. For ranking scores we decided to use the probability of class *true* to be our score. Furthermore, considering the huge success that gradient boosting approaches had in competitions such as ones held at Kaggle[2] we experimented with LightGBM framework which ultimately proved itself to be a better choice for the tasks.

3.1. LightGBM

LightGBM framework is an implementation of the gradient boosting decision tree algorithm. It differs from other tree-based algorithms by growing the tree vertically (leaf-wise), while other algorithms grow the tree horizontally (level-wise). By doing this, it is able to reduce more loss than level-wise algorithms and results in a better accuracy. Inside LightGBM we opted for their ranker model which uses *LambdaRank* (Burges et al., 2007) algorithm which deals with optimizing the quality measures by using implicit cost functions. It does this to avoid the problem of cost derivatives being zero or undefined. As official metric is MAP score, we used the same metric inside LightGBM. The optimal values of LightGBM Ranker parameters that we found by 10-fold cross validation and are different from defaults are:

- max_bin (subtask A: 80, subtask B: 140, default: 255): Maximum number of bins that feature values will be bucketed in. Smaller number may reduce accuracy, but increases general power (reduces overfitting).
- $min\_data\_in\_leaf$ (subtask A: 20, subtask B: 30, default: 20): Minimal number of data in a leaf.
- num_leaves (subtask A: 11, subtask B: 8, default: 31): Maximum number of leaves in one tree.
- $min\_gain\_to\_split$ (subtask A: 2, subtask B: 2, default: 0): Minimal gain to perform the splitting of the tree.

Additionally, we tried LightGBM Regressor which outputs the probability of class *true* and that probability was used as a ranking score. Inside that model we could not use MAP score anymore, but binary logistic loss. We optimized the same parameters as for LightGBM Ranker and obtained them the same way.

3.2. Preprocessing

For preprocessing in both tasks we used Stanford Core NLP to obtain part-of-speech tags (POS-tags), named entities (NE) and dependency parse trees which were used later for calculating numerical features. Also, we used NLTK (Bird et al., 2009) separately for tokenizing and stopword removal.

[2] https://www.kaggle.com/

Table 1: Subtask A results.

Model	MAP	AvgRec	MRR	P	R	F1	Acc
Random baseline	62.30	70.56	68.74	53.15	75.97	62.64	52.70
IR	72.61	79.32	82.37	-	-	-	-
KeLp-primary	88.42	93.79	92.82	87.30	58.24	69.87	73.89
Logistic regression	80.37	87.78	86.02	-	-	-	48.02
LightGBM LambdaRank	81.24	88.15	88.25	-	-	-	48.02
LightGBM Regressor	80.10	87.30	86.75	-	-	-	48.02

Table 2: Subtask B results.

Model	MAP	AvgRec	MRR	P	R	F1	Acc
Random baseline	29.81	62.65	33.02	18.72	75.46	30.00	34.77
IR	41.85	77.59	46.42	-	-	-	-
KeLP-contrastive1	49.00	83.92	52.41	36.18	88.34	51.34	68.98
Logistic regression	41.01	74.96	45.21	-	-	-	81.48
LightGBM LambdaRank	44.11	78.94	46.96	-	-	-	81.48
LightGBM Regressor	45.21	78.63	49.22	32.92	80.98	46.81	65.91

3.3. Ngram Overlap Features

Some of the most common features used in previous works are based on unigram, bigram and trigram overlap. Let S_1 and S_2 be sets of succeeding n-grams in the first and second sentence that we wish to compare. We then calculate the overlap as:

$$NGramOverlap(S_1, S_2) = 2 \cdot \left(\frac{|S_1|}{|S_1 \cap S_2|} + \frac{|S_2|}{|S_1 \cap S_2|} \right)^{-1} \quad (1)$$

Put simply, n-gram overlap is computed as a harmonic mean of values representing the degree to which first sentence has in common with second sentence and vice versa. The overlap formula, defined by (1), is used to compute the overlap for unigrams, bigrams and trigrams, each being a separate feature.

Additionally, we include skip-gram overlap features, namely skip-bigram and skip-trigram overlap. These help us model the possibility of n-gram words not appearing one after another, as there can be arbitrary gaps between them. We chose to allow gaps of 2 words, meaning that we can skip 2 words and use the third word. We experimented with larger gaps, but score was slightly decreasing as we expanded the gap. We calculate the skip-gram overlap by using equation (1) with the difference that S_1 and S_2 represent sets of skip-grams.

3.4. WordNet-Augmented Word Overlap

Since unigram word overlap depends on the words from two different sentences to be the same, we would like to have a certain amount of leeway for lexical variation. Hence, we take advantage of WordNet for coming up with a score for words that do not appear in both sentences. We use POS-tags to query WordNet using NLTK. First, we try to get the first synset returned using both token and tag from POS-tag. If we fail, then we try to retrieve a synset using only token. WordNet augmented coverage $P_{WN}(\cdot, \cdot)$ is defined as follows:

$$P_{WN}(S_1, S_2) = \frac{1}{|S_2|} \sum_{w_1 \in S_1} score(w_1, S_2) \quad (2)$$

$$score(w, S) = \begin{cases} 1 & \text{if } w \in S \\ \max\limits_{w' \in S} sim(w, w') & \text{otherwise} \end{cases} \quad (3)$$

where $sim(\cdot, \cdot)$ represents the WordNet path length similarity (Šarić et al., 2012). Finally, the value used as a feature is a harmonic mean of $P_{WN}(S_1, S_2)$ and $P_{WN}(S_2, S_1)$.

3.5. Named Entity Overlap

Another feature we used is a simple named entities overlap. We decided to go for the straightforward approach of comparing how many named entities of any type such as "ORGANIZATION", "PERSON" or "LOCATION", are there in both sentences, i.e., we sum the number of named entity overlaps. Each named entity found in the sentence is saved as a tuple $(word, type)$ because it is important when calculating the overlap that both words are of the same type. We then calculate the value that is used as feature in our system as:

$$P_{NE}(S_1, S_2) = 2 \cdot \left(\frac{countNE(S_1)}{overlap(S_1, S_2)} + \frac{countNE(S_2)}{overlap(S_1, S_2)} \right)^{-1}$$

where $countNE(\cdot)$ represents the number of named entities in the sentence and $overlap(\cdot, \cdot)$ represents the number of same named entities, namely tuples representing named entities, in both sentences.

3.6. Tf-Idf Features

Additionally, we compute *Tf-Idf* vectors in both subtasks. For subtask A, we compute *Tf-Idf* vectors between: question subject and comment text, question body and comment text, and together question subject and body and comment text. For subtask B, we compute vectors between: original and related question subject, original and related question body, original question subject and related question body and vice versa, and original and related question subject and body together. Then we calculate the cosine similarity between the vector pairs and include those values as numerical features for our model.

3.7. WordNet Lin Similarity

Using the same POS-tags and the same querying technique described in section 3.4., we compute how much comment text is similar to subject and body together and vice versa. We used Lin similarity which is based on the information content. Information content values were obtained using NLTK's already computed information content from Brown corpus. Values were calculated using equations (2) and (3).

3.8. Dependency Types Overlap

As a result of dependency parsing, we got tuples formed as $(dependencytype, governinglemma, dependentlemma)$. Overlap was simply calculated as a Jaccard similarity using equation (4) between question subject and body, and comment for subtask B, and original and related question subject and body together.

$$J(S1, S2) = \frac{|S1 \cap S2|}{|S1| + |S2| - |S1 \cap S2|} \quad (4)$$

4. Results

For our experiments, we used official datasets which were already split as follows: (i) train dataset, (ii) development dataset and (iii) test dataset. We used train set inside 10-fold cross validation to tune hyperparameters and additionally we used development dataset in LightGBM fit method so that we prevent overfitting for each set of hyperparameters. MAP score was used as cross validation score.

Our results are shown in Table 1 for subtask A, and in Table 2 for subtask B. Among our 3 models, LightGBM LambdaRank gave the best results for subtask A, while LightGBM Regressor has shown itself to be the best fit for subtask B. Additionally, we show the scores of the best system submitted to the competition.

4.1. Significance Testing

We use the paired sample *t*-test to test whether the LightGBM model is a statistically significant improvement in MAP score over logistic regression model. The two implementations are trained using the exact same features on both subtasks. We get a p-value of 0.18 (subtask A) and 0.051 (subtask B), which is more than the cutoff threshold. Therefore, we can not reject the null hypothesis, and can not conclude that either of models performed better, given the same feature set.

5. Future Work

Future work would be exploring more features based on semantic text similarity, but also exploring simple features from question and comment, e.g., sentence length. Additionally, since we tested only on logistic regression and LightGBM, it would be interesting to try different models and see the results.

6. Conclusion

We presented a LightGBM approach to (A) rank comments according to relevance to a given question, and (B) rank already existing questions according to relevance to a given question. We also train a logistic regression model on the same feature set. Results show that LightGBM, even though shown to be very competitive in previous work, has not performed significantly better than simple logistic regression given these features.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*, pages 193–200.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3149–3157.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

Preslav Nakov, Lluís Màrquez, Walid Magdy, Alessandro Moschitti, Jim Glass, and Bilal Randeree. 2015. Semeval-2015 task 3: Answer selection in community question answering. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 269–281.

Preslav Nakov, Doris Hoogeveen, Lluís Màrquez, Alessandro Moschitti, Hamdy Mubarak, Timothy Baldwin, and Karin Verspoor. 2017. SemEval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, SemEval '17, Vancouver, Canada, August. Association for Computational Linguistics.

Deepak Ravichandran and Eduard Hovy. 2002. Learning surface text patterns for a question answering system. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 41–47. Association for Computational Linguistics.

Filip Šaina, Toni Kukurin, Lukrecija Puljić, Mladen Karan, and Jan Šnajder. 2017. Takelab-qa at semeval-2017 task 3: Classification experiments for answer retrieval in community qa. In *Proceedings of the 11th Inter-*

national Workshop on Semantic Evaluation (SemEval-2017), pages 339–343.

Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. 2012. Takelab: Systems for measuring semantic text similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 441–448. Association for Computational Linguistics.

Quan Hung Tran, Vu Tran, Tu Vu, Minh Nguyen, and Son Bao Pham. 2015. Jaist: Combining multiple features for answer selection in community question answering. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 215–219.

Toxic Language Detection in Wikipedia's Talk Page Edits

Lucija Belić, Zvonimir Cikojević, Jasmin Redžepović

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{lucija.belic,zvonimir.cikojevic,jasmin.redzepovic}@fer.hr

Abstract

In order to increase the quality of user experience in using social networks it is important to keep the toxic and foul language at a marginal level, preferably non-existent. It is quite difficult, if not impossible, to achieve this using only human administration that filters the aforementioned type of language. Another solution is to automatize the process of toxic language detection using machine learning models. In this paper we propose such solution and show its promising results.

1. Introduction

In today's world more and more people use internet as a means of communication. A common type of communication are comments on social networks. Even though undesirable, it is certainly reasonable to expect that there will be foul language present in communication. Therefore the question is how to restrain such language and provide a safer and more peaceful online environment. One way to tackle this is to set up a human administration layer to filter the foul language. It is quite obvious this will not scale well in large systems. A better way is to automate the process of foul language detection.

In this paper we give an overview of a supervised machine learning model used to tackle the task of toxic language detection. We first describe the dataset used to train the models: comments from Wikipedia's talk page edits and address the issues that arise from using it. In subsequent sections we present machine learning models and their performance evaluation.

2. Related Work

Since this task is taken from *Kaggle* competition: Thain et al. (2017), it is quite clear many attempts have been made and have proven to be successful. Top leaderboard competitors have reached such high performances that one might conclude this task to be solved. There have been other similar researches regarding detecting abusive and offensive language, such as Sax (2016) and Kennedy et al. (2017) Pedregosa et al. (2011). To our knowledge, there has not been provided solution that combines different models exactly the same way we do in this paper.

3. Dataset Overview

Dataset we used is a collection of comments from Wikipedia's talk page edits, Thain et al. (2017). Comments that are considered to be clean have no associated label. Every other comment is labeled into one or more labels: *toxic*, *severe_toxic*, *obscene*, *threat*, *insult*, *identity_hate*. A sample of comments is shown in Table 1. More specifically, the first comment labeled as *clean* and the second comment labeled as *toxic*, *severe_toxic*, *obscene* and *insult*.

4. Data Preprocessing

In this section we shall firstly present the NLP pipeline used to preprocess the comments. Secondly, we present class imbalances that are contained in the training set and our solution for it.

4.1. NLP Pipeline

Our pipeline consists of three components all taken from *Scikit-learn* library (Pedregosa et al. (2011)).

1. *TfidfVectorizer*
2. *TruncatedSVD*
3. *Normalizer*

The first step is to convert raw text of comments into a matrix of TF-IDF features. The next step is to perform linear dimensionality reduction by means of truncated singular value decomposition. Finally, each sample with at least one non-zero component is rescaled independently of other samples so that its norm equals one.

4.2. Class Imbalances

A comment is considered to be clean if none of the labels are present. Total number of those comments is equal to 143346. The total number of all other comments with at least one label is equal to 16225. It is clear that a high class imbalance between clean and toxic comments is present. Furthermore, when we examine the class distribution among toxic comments, class imbalance is present as well, which is shown in Figure 1.

First, we shall describe the process of removing class imbalances present in toxic comments, subsequently the process of removing class imbalances between clean comments and **balanced** toxic comments.

In order to remove imbalances present in toxic comments, a somewhat heuristic approach was chosen. Comments that have labels with counts less than 0.5 standard deviation were oversampled. On the other hand, those comments with labels' counts above 0.5 standard deviation were undersampled. Figure 2 presents this idea in a clearer fashion.

Comments with labels *identity_hate*, *severe_toxic* and *threat* were oversampled, while comments with label *toxic* were undersampled. Finally, comments with labels *insult* and *obscene* were left as they were. Results of this method are shown in Figure 3.

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 20–23, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Table 1: Dataset sample: one clean comment and one toxic comment with their respective labels

comment	toxic	severe_toxic	obscene	threat	insult	identity_hate
You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
COC****ER BEFORE YOU PI*S AROUND ON MY WORK...	1	1	1	0	1	0

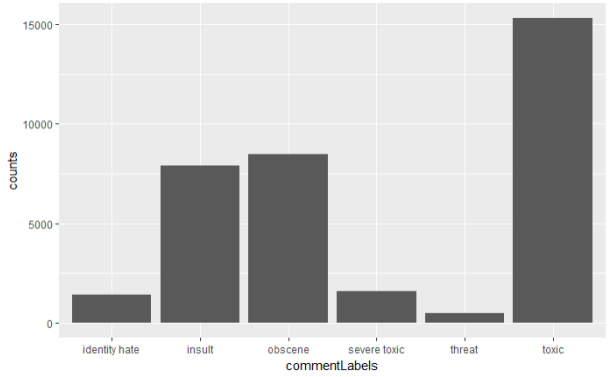


Figure 1: Toxic comments labels distribution

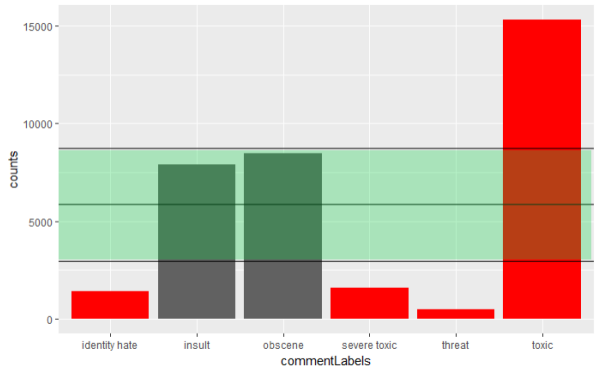


Figure 2: Toxic comments labels distribution with a mean line. Highlighted area represents 0.5 standard deviation from the mean value.

It is important to mention there is a flaw in this approach. The nature of this issue arises from the fact that this is a multilabel problem, meaning there can be multiple labels per comment. From the Figure 2 we can clearly see class *threat* needs to be resampled. If we do that, it is expected that resampled comments are not labeled solely as *threat*. Therefore, count for other labels will rise as well. Consequently, it is quite difficult to reach a uniform distribution of comments per class, however Figure 3 shows this method is still quite robust.

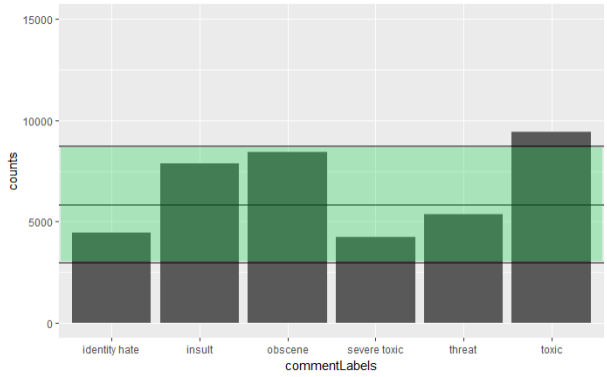


Figure 3: A more uniform toxic comments labels distribution

5. Methods

In this section we start off with the standard multilabel classification. In the following subsections we divide the problem into two subproblems: one for dealing with discriminating between clean and toxic comments, other for discriminating only between toxic comments (*toxic*, *severe_toxic*, *obscene*, *threat*, *identity_hate*, *insult*). We combine solutions for these two subproblems into the final joint model output.

5.1. Multilabel Classification Approach

Each comment can be labeled as one or more of the following: *toxic*, *severe_toxic*, *obscene*, *threat*, *identity_hate*, *insult*. If however, no label is provided for the comment, it is considered as clean.

5.2. Multilabel Classification Model

For this particular problem, we used one-vs-rest strategy implemented in *Scikit-learn* package - Pedregosa et al. (2011). The models applied with this strategy are: logistic regression, naive bayes, decision tree. In section Experiments we discuss the setup for these models, hyperparameter tuning and their results.

5.3. Joint Classification Approach

Our joint model solution consists of two components:

1. binary classification
2. multilabel classification

We first train the binary model to learn how to differ toxic comments from clean comments. Next we train the multilabel model to learn how to differ toxic comments only. To

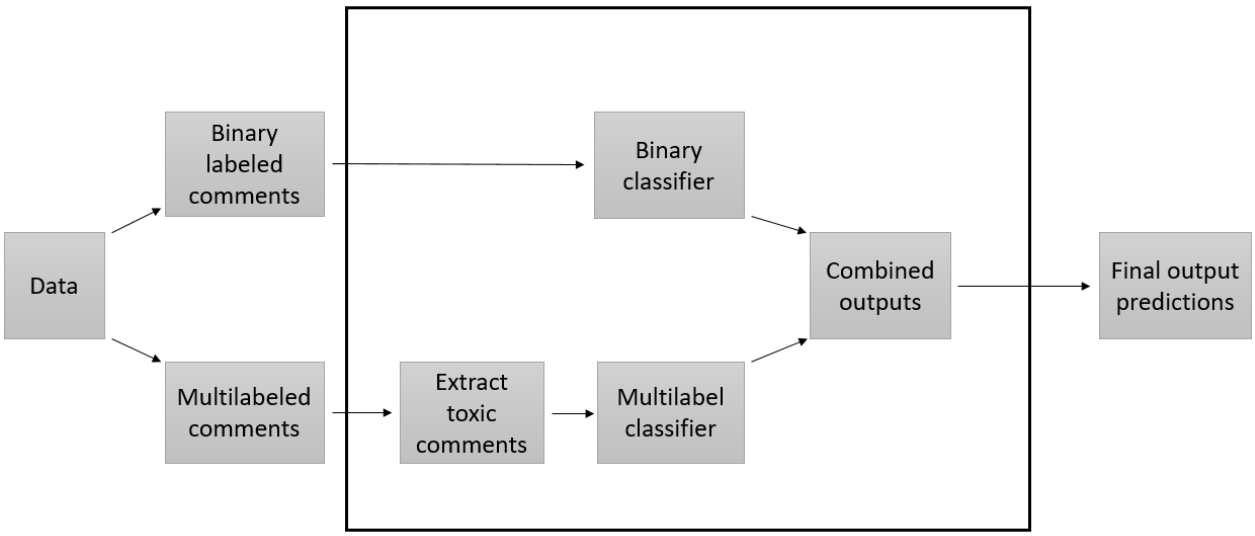


Figure 4: Each component in the joint model receives the necessary data and their predictions are then combined into a final output prediction.

combine the predictions from these two components, we do the following:

1. all the comments the binary component predicted to be clean are labeled as clean
2. all the comments the binary component predicted to be toxic are labeled as toxic
3. all the binary toxic labels are expanded into multilabels by the multilabel component.

For a clear illustration, see Figure 4.

5.4. Binary Classification Approach

One would probably agree that presented classes are not disjunctive. In order to reduce the problem complexity, instead of having to deal with 6 similar labels, we present two very distinct labels: toxic and clean. Any comment that has at least one label is considered toxic, whereas all comments that have no associated label are considered clean. This is the core idea of the first component in our joint model solution.

5.5. Binary Classification Model

The models applied with this strategy are: logistic regression, naive bayes, decision tree. In section Experiments we discuss the setup for these models, hyperparameter tuning and their results.

5.6. Specialized Multilabel Approach

This section describes the second component of our joint model solution.

The difference between this specialized approach and the one described in Sections 5.2. and 5.1. is the type of comments used to train the models. Models in this approach are trained solely on toxic comments, hopefully enabling them to better distinguish toxic labels.

Table 2: Experiments Results

	Macro average	Micro average
Mutlilabel approach	0.8069	0.8485
Joint model approach	0.8193	0.8731
Top *Kaggle* competitor	0.9885	-

6. Experiments

All the tests were run on a test set taken from the *Kaggle* competition Thain et al. (2017). Hyperparameter tuning for all models was done using 5-fold cross validation. Here is a full list of machine learning models we used in all approaches:

- Logistic regression
- Naive Bayes classifier
- Decision tree
- Random forests

In this paper, however, we only show results of the best performing ones. A measure used to evaluate performance was set by *Kaggle* competitions administration, which is mean columnwise area under receiver operating characteristic curve, therefore this is the metric we used to evaluate our models as well. We can see from the Table 2 that micro average ROC AUC score is greater than the macro average score, indicating that the model fails to properly classify classes with relatively few samples. For both multilabel approach and joint model approach the best performing model was the simple logistic regression with regularization parameter C set to 5.

7. Conclusion

From the perspective of social networks, automated toxic language detection is still a high priority issue that is constantly being improved in order to have a stable and peaceful environment for online social interaction.

In this paper we've presented our solution to toxic language detection that provided arguably promising results. Our models' performance were compared to the top *Kaggle* competitors and have fallen quite behind. Nevertheless, we believe our joint model approach could reach top performance when combined with LSTM models, which is the base of our future work. A simple statistical significance test to see whether there is any improvement in using joint model over the standard one is also left for future work due to shortage of time.

References

George Kennedy, Andrew McCollough, Edward Dixon, Alexei Bastidas, John Ryan, Chris Loo, and Saurav Sahay. 2017. Technology solutions to combat online harassment. pages 73–77.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Sasha Sax. 2016. Flame wars: Automatic insult detection.

Nithum Thain, Lucas Dixon, and Ellery Wulczyn. 2017. Wikipedia talk labels: Toxicity. Feb.

Depression Detection From Language Use

Ana Bertić, Roko Srđan Buča, Tvrtko Sternak

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ana.bertic,roko-srdan.buca,tvrtko.sternak}@fer.hr

Abstract

Depression detection is a burning problem in today's age of technology. Studies show that it is possible to predict the person's mental state by examining their language use. On social media there is an abundance of written text by its users which can be harvested and used to identify individuals at risk of depression. In this paper, it is shown that with our methods of careful feature selection and class imbalance handling, our deep neural network performs above current state-of-the-art algorithms when trained on research collection of Reddit posts. We also proved that despite the fact that simpler models obtain promising results when using our feature extraction, deep models nevertheless exceed their performance pushing the F1 score to 0.67 from 0.64.

1. Introduction

The World Health Organization estimates that more than 300 million people worldwide are now living with depression, which is 18 percent more than 2005. Since depression is often underdiagnosed there has always been a need to seek effective strategies for early detection, intervention and appropriate treatment of diagnosed individuals. It has been shown that machine learning algorithms can solve the problem of detecting early signs of depression from language use. Our work concerns improvement and validation of possible models, as well as finding new and untried ones that can be used for this purpose. The datataset used for training and testing our approach is a test collection for research on depression and language use (Losada and Crestani, 2016). Dataset is composed of 892 Reddit users, precisely documents containing their post histories which are binary classified in depression and control groups.

Data preprocessing in this paper consists of standard POS tagging and lemmatization techniques followed by four types of feature extractions. Three bag-of-words (BoW) models were created with an additional extracted and vectorized "special" features (Section 3.1.).

Our deep neural network (DNN) model which is the focus of this paper uses feature selection based on chi2 word scores, simple oversampling techniques seen in Section 4. and additional features which were shown to reliably predict early depression signs (Wang et al., 2013). DNN uses fixed length feature vectors where each vector represents an entire post history of a given user.

Additionally, we compare our DNN model to itself when various feature selection methods are used, as well as with previous work in this field and ensambles of simpler machine learning models. We show that it is possible to outperform current state-of-the-art models with ensambles which use our feature extraction techniques.

2. Related Work

Even though in the age of social media a massive amount of information can be extracted from social networks, for a long time there has been a lack of publicly available data for doing research on detecting language patterns associated with depression. Prior researches usually included analyses from microblogs like Twitter, proving that large platforms reflect the expression of depression both explicitly and implicitly.

However, in case of Twitter, the limit of 280 characters is too small to provide any meaningful context. We used collection from open-source platform Reddit carefully constructed by (Losada and Crestani, 2016) where each subject is represented via document of a large number of posts and comments submitted during a longer period.

Numerous text mining techniques have been developed based on background knowledge of psychologycal reasearches (Wang et al., 2013), some of them including sentiment analysis, linguistic rules and vocabulary construction. However, there has still not been substantially significant improvement of results in conducted related work and existing tehniques generally rely on experimenting with baseline models in hope of achieving better results which rarely exceed over 60% F1 score.

For sentiment analysis a lexicon and rule-based sentiment analysis library VADER (Gilbert, 2014) is used. It is found to work well when integrated into machine learning models and easily handles text originated in social media. VADER is also used in (Leiva Aranda, 2017) representing additional features during data preprocessing. It contains methods which retrieve percentage of negative, positive and neutral sentiment representative words and calculates overall sentence sentiment polarity.

In this paper we, use machine learning models and ensembles similar to that of (Polikar, 2006) in order to test our feature extraction independent of our modeling methods. Based on the success of our feature extraction methods we developed a DNN which showed improved performance when using those extracted feature vectors.

In our final testing, we compare our model to the best model presented in an overview of a competition (Losada et al., 2017) based on the dataset we used.

3. Feature Evaluation

Corpus constructed from words used by subjects in training set consisted of more than 120.000 distinct words which are mostly typos, links and numbers that don't have any dis-

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 24–27, University of Zagreb,
Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

criminatory interpretation. With these observations, it was necessary that the first step in our approach to depression classification is feature evaluation and extraction.

We decided to observe results based on three approaches. Effectiveness of each approach on the validation set is shown in Table 3.

• **augmented frequency interval feature selection:** Every lemmatized word in the corpus was counted and several word intervals based on word frequency were chosen for testing. Three sizes of word vectors were used: 200, 400 and 1500. We found that models that used word vectors bigger than 200 were prone to overfitting, even with significant dropout added to neural network layers. Frequency interval that proved to work best was the one that contained words with in-corpus frequency being between 6.5% and 3%. Chosen word vector was then augmented by removing numbers and other non-words which were then replaced with words that proved to be of value in previous model selection testing.

• **term frequency-inverse document frequency feature selection:** For each word a tf-idf weight was calculated. This value represents the importance of a word in a document, considering whole corpus. Based on weights, tf-idf matrix was constructed in which each row represented a tf-idf value for each word in corpus for a single document. The matrix rows were labeled into a document of depressed and non depressed user and passed to a random forrest classifier in order to extract most important features, for this classification. 247 words were extracted and filtered down to 200 which were used in further classification.

• **Chi2 feature selection:** After extraction of BOW representation for each user best 150 words were determined using chi2 algorithm based on χ^2 statistic (Liu and Setiono, 1995). The limit of 150 words was chosen based on observation of words that appeared when the limit increased, most of those words were typos or numbers and did not represent any objective semantic meaning. Chi2-based feature selection produced models with best generalization properties and was for that reason used by our DNN model, especially when augmented with additional features.

3.1. Special Data

Inspired by the work of (Wang et al., 2013), we did a similar analysis of the corpus in order to find additional features with discriminatory value.

Much like previous works, we found that the use of emoticons was a powerful predictor of somebody's mental state, especially and unsurprisingly the use of negative emoticons. We managed to recreate the general gist of the (Wang et al., 2013) and also found that use of personal pronouns in either singular or plural form can be used as another feature for detecting early signs of depression.

There were some other factors used which can be found in Table 1. Even though we ran our statistical evaluation on both the train set and test set and found some differences, we modeled our special feature vectors based only on the evaluation of the train data. Additionally, it is important to point out that some models had more trouble than others with using this extra feature vector. Our final DNN presented in this paper does not use all nine features, but instead uses the best four features based on the train data evaluation. We chose specifically four features based on both the empirical experimentation and the obvious gap between the fifth and sixth most relevant feature. We didn't use the 'emoticon use' feature since that knowledge is already contained in other used emoticon statistics.

4. Oversampling Method

To remedy the problem of severe class imbalances in used dataset (ratio of depressed/control groups is around 1/5), we used various oversampling methods.

In our baseline model we used Synthetic Minority Oversampling Technique (SMOTE) proposed in (Chawla et al., 2002). SMOTE generates synthetic minority class instances between existing minority instances in euclidean space. Final experiment presented in Table 4 shows that this approach significantly improves models F1 score on the test set.

Since DNN uses an extra special feature vector which contains different diferent type of data and additonally expand the dimensionality of the input vector we resorted to using other, simpler, oversampling technique. Used techinque artifically increases weights of depressed subjects by copying positive training examples five times in order to equalize the dataset imbalance.

5. Classifier Model

This paper presents two different and tested improvements to previous solutions, especially in regards to (Losada and Crestani, 2016) since our proposition shares the most similarity in approaching the problem of detecting early signs of depression.

We propose a user-level representation of data based on BoW models for the first step of the model improvement, we do not use special features in this step. We then apply models and ensambles similar to those used in previous works. To make it more fair, we separately train those machine learning models in order to achieve the best possible results on the feature vector types we use.

For the second step of the model improvement, we present the best DNN we constructed for this problem. The testing showed that the feature complexity can be much better described with the complex function DNN provides than with simpler machine learning algorithms. The final DNN achieves 0.67 F1 score compared to 0.65 of our ensamble. However our calculations show that this improvement is not statistically significant. Both the DNN and the ensamble work best when using the chi2-based BoW vectors.

5.1. Baseline Classifier

After evaluation of the dataset and its size, we concluded that the best way to tackle the problem of depression classification is by using an ensamble of simple classifiers. The ensamble consists of five individually fine tuned classifiers on train set. In Table 2 we justify the usage of each classifier in the final ensamble by observing that the F1 score

Table 1: Special features statistics on train set (features used in our model are bolded)

Special feature (average)	Depressed users	Non-depressed users
emoticon use	26.7952	13.87096
positive emoticon use	20.1084	10.8064
negative emoticon use	6.6506	3.0074
sentence length	17.8904	20.3568
first person pronoun use	0.04573	0.024899
plural first person pronoun use	0.00266	0.00308
posting time	23:39:09	22:47:41
number of posts between 00:00 and 06:00	30.45%	27.46%
number of posts between 09:00 and 16:00	17.39%	21.06%

on 5 fold cross validation with chi2 feature selection of ensamble is highest when all five classifiers are used.

Models used in ensamble and their hiperparameters were:

• **Logistic regression:** Regularization was done with L2 penalty, and regularization strength λ was set to 20. We didn't use oversampling when validating the classifier so class weight parameter was set to be directly proportional to class frequencies in the input data ('balanced').

• **Random forest**

• **SVM:** C-support vector classifier with rbf kernel. Penalty parameter of the error term was set to 0.35 and class weight parameter was also set to 'balanced' like in logistic regression.

• **Ridge classifier:** Regularization strength α was set to 2.5

• **Ada boost:** Number of classifiers was set to 150 and learning rate to 1.

Table 2: ensamble f1 scores on validation with leaving one classifier out on each validation

Ensamble	F1 validation score
w/o logistic regression	0.57
w/o random forest	0.61
w/o SVM	0.70
w/o ridge	0.70
w/o ada boost	0.59
full ensamble	0.72

5.2. Deep Neural Network

DNN presented in this paper is a fairly simple model. It has four hidden layers which use the ReLU activation function. With all data already preprocessed and loaded into RAM, it takes only a couple of seconds to converge when run on a standard Intel i5 processor. Model has probabilistic output as defined by two neurons with softmax activation function in the last layer. To determine the class a datapoint belongs in we simply use an argmax function.

We tuned all hyperparameters by doing an extensive search. 10-fold validation setup was used to determine the best models. During optimization, we only used F1 score as the accuracy score is not very useful due to the inability to detect overfitting on the imbalanced dataset. Even though we used SMOTE oversampling while training the baseline, we found that a simple artificial data weighing worked better for the DNN.

Several different approaches were attempted and compared. The bare DNN that uses only the main BoW vector represented dataset performs similarly to the ensamble with a F1 score of 0.63. In order to additionally improve our models we implemented additional features as described in the section 3.1. in the DNN. Best DNN constructed uses only the top five most significant features from the special feature vector. DNN performs better when the special feature vector is concatenated to the BoW vector and fed to the network, rather when the cascade of classifiers is used. This shows that there is an implicit interaction between chi2-based BoW model and the special features which the ensamble of classifiers cannot fully encompass.

6. Experiments

Table 3: ensamble F1 scores on validation with preprocessed data using feature selection methods

Feature extraction method	F1 validation score
w/o feature selection	0.69
w/ afi feature selection	0.7
w/ tf-idf feature selection	0.75
w/ chi2 feature selection	0.72

In our final experiment we compare F1 scores of our proposed models to those from previous works on the standard test set. Models that we compare ours to are the proposed baseline from (Losada and Crestani, 2016), model proposed in (Leiva Aranda, 2017) and the current state-of-the-art FHDOA model proposed in (Losada et al., 2017). We show in Table 4 that both of our models outperform all other models F1 score-wise (0.65 and 0.67 compared to 0.64 achieved by FHDOA). We also point out that our neural network with

Table 4: Final results on test sets comparing to (Leiva Aranda, 2017),(Losada and Crestani, 2016) and (Losada et al., 2017)

Model	p	r	f1
Genetic Algorithm + VADER	0.45	**0.77**	0.57
Logistic regression	0.64	0.59	0.62
FHDOA	0.61	0.67	0.64
Ensamble (afi)	**0.76**	0.42	0.55
Ensamble (tf-idf)	0.73	0.46	0.57
Ensamble (chi2)	0.75	0.56	0.65
Ensamble (chi2) w/o oversampling	0.49	0.74	0.37
Bare neural networks (chi2)	0.65	0.61	0.63
Augmented neural networks (chi2)	0.65	0.68	**0.67**

augmented feature vectors has the most balanced precision-recall score.

Without using special features our ensamble performs better than our neural network. This is probably due to relatively small train set size. However, when we add special features the ensamble suddenly performs worse while the neural network significantly improves it's F1 score. We conclude that this type of augmented feature vector contains a high level of feature interactivity that is better suited for deeper models.

7. Future work

Various improvements could be done to boost the presented model. With the help of psyclogical and deeper statistical research, more special features could be added to the system. It would also be beneficial to include sentiment polarity as an additional feature. More complex models could be created that are compatible with a more fine-grained way of data representation instead of the used coarse user-level one.

Given that our models are easily overfitted on the available data it is probable that simply adding more data would automatically improve models' performance.

8. Conclusion

In this work we experimented with both machine learning and deep learning models in hope to design a model that is effective at recognizing signs of depression in Reddit users based on their post histories. Starting with ensembles of simple classifiers, we combined three different feature selection methods and studied the results. Models that use Chi2 feature selection proved to be the most successful in this task, surpassing other state-of-the-art models' F1 score by a significant margin. Additionally, we focus on one particular DNN model which reliably outperforms ensambles aforementioned.

The model is augmented with additional text-mined features which were shown to contain significant linguistic markers in related works. Our results have proven that the classification of depressed users is possible with deep models as well as with ensambles of simple models, both of which outperform current state-of-the-art techniques when using the feature extraction presented in this paper.

9. Acknowledgments

We would like to use this opportunity to thank our professor and mentor, Sc.D. Jan Šnajder for the invaluable knowledge of this field that he has provided us. We would also like to than Sc.D. Mladen Karan for suggesting the chi2 feature evalution and for showing us how to perform statistical significance testing.

References

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

CJ Hutto Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf.*

Víctor Leiva Aranda. 2017. Towards suicide prevention: early detection of depression on social media.

Huan Liu and Rudy Setiono. 1995. Chi2: Feature selection and discretization of numeric attributes. In *Tools with artificial intelligence, 1995. proceedings., seventh international conference on*, pages 388–391. IEEE.

David E Losada and Fabio Crestani. 2016. A test collection for research on depression and language use. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 28–39. Springer.

David E Losada, Fabio Crestani, and Javier Parapar. 2017. Clef 2017 erisk overview: Early risk prediction on the internet: Experimental foundations.

Robi Polikar. 2006. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45.

Xinyu Wang, Chunhong Zhang, Yang Ji, Li Sun, Leijia Wu, and Zhana Bao. 2013. A depression detection model based on sentiment analysis in micro-blog social network. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 201–213. Springer.

Personalized Medicine: Redefining Cancer Treatment classification using Bidirectional Recurrent Convolutions

Mihaela Bošnjak, Grgur Kovač, Franko Šesto

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{grgur.kovac,franko.sesto}@fer.hr, mihaella.bosnjak@protonmail.com

Abstract

In this paper we propose a method for representing documents as fixed-size representations. The method consists of two parts: the tf-idf retrieval part that retrieves a predetermined number of most relevant sentences and a layer we named a Bidirectional Recurrent Convolutional layer (BRC). The BRC layer is a combination of a bidirectional recurrent unit and a convolution. For every sentence, it computes a fixed size vector that contains the information from both the context and the sentence. The output is thus fully fixed so it can be fed into a fully connected layer for classification. The model is evaluated on kaggle's dataset for the task "Redefining Cancer Treatment".

1. Introduction

One of the problems we face while analyzing texts is their variation in length. Getting from a variable size length to a fixed vector that represents text and preserves important information presents a great problem. In this work we propose a novel way of dealing with variable length texts while focusing on the preservation of context.

We have applied this idea on a Kaggle's competition task "Redefining Cancer Treatment". In this task for each gene-mutation pair we are given a corresponding document that talks about them. We have decided to use that gene-mutation information as a query for important information extraction from the document. And then we are to classify that gene-mutation pair as a driver or passenger based on the given document.

The given dataset greatly varies in size which is an appropriate task for our BRC layer to tackle. One of the main contributions of this paper is a BRC layer that represents variant length sentences as fixed-sized vectors. The second one is our pipeline approach. As we are working with large bodies of text, some of the sentences hold greater value then others. We have decided to use a *td-idf* retriever to help us track down important sentences. This task also requires from us to classify some document with regard to a query. Our approach with *td-idf* comes in handy as we can single out sentences that contain information that is relevant to the given query. In combination with our BRC layer we present a pipeline for tracking important information while preserving context and mapping sentences into a fixed-size vector.

2. Related Work

In recent years deep learning has been applied to many natural language processing problems. One often used component is an LSTM introduced by Hochreiter and Schmidhuber (1997). Also, CNNs have become quite popular, primarily in computer vision (Krizhevsky et al., 2012; Karpathy and Fei-Fei, 2015), and also in language processing (Kim, 2014; Zeng et al., 2014).

To the best of our knowledge there are no published papers on this specific Kaggle task yet. We have decided to frame this problem as a large scale relation classification task. One popular approach of relation classification is using convolutional neural networks (Sunil Kumar Sahu, 2016; Ji Young Lee, 2017). Ji Young Lee (2017) also used rule-based post-processing to correct relations detected by CNN. Yan Xu (2016) proposed a deep recurrent neural networks for the same task of relation classification. They showed that different layers learn different representations.

The big difference between those tasks and ours is that they do relation classification from a single sentence while we do relation classification from a document containing a large number (up to 5000) of sentences.

It is also worth mentioning that we take a lot of inspiration from (Yang Yu, 2016) paper that deals with the task of question answering. They use Bi-GRUs to represent variant length sequences as fixed length representation in the same manner that we use Bi-LSTMs to represent sentences. They also proposed a *dynamic chunk reader* that is able to extract and rank a set of answer candidates of variable lengths from a given document.

3. Redefining Cancer Treatment

Redefining Cancer treatment is 2017 *Kaggel's* competition. Its main goal is automated classification of genetic mutations based on their contribution to tumor growth.

The dataset contains clinical papers as input and a file containing additional information. In the additional file we are given a gene, its mutation, the id of the clinical text we are using and the ending label. Labels span from 1 to 9 representing different types of malignancy of mutations on specific genes.

The train set contains 3320 examples. Each example contains an inquiry about a gene-mutation pair and a specific paper to look at. Each paper talks not only about one gene and mutation, but it also covers a range of different mutations and additional information. The number of unique clinical papers for the training set is 1921. There are two test sets. The first one, named stage-1, contains 5668 examples. It is bigger than the training set as there are

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 28–32, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

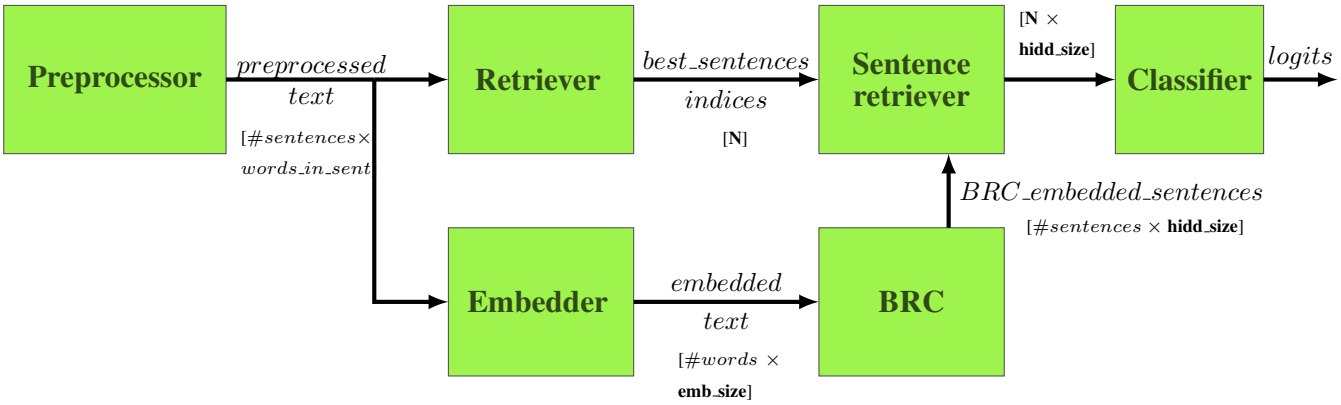


Figure 1: The pipeline of our model separated into six components. Fixed dimensions are made bold.

computer generated examples, so that any kind of cheating is prevented. Dataset cleared of generated examples contains only 368 examples and the second test set, named stage-2, contains 986 examples and is further split into public and private subsets.

4. Our Approach

Now we will briefly describe the relevant parts of our model. It can be divided into 6 separate components as shown in Figure 1.

The preprocessor part does basic text operations that include pos tagging, lemmatization, stop words removal and so on, which are described in 4.1.

The preprocessed text is then passed onto two components. The Retriever component calculates *tf-idf* values for each **sentence** and the query. Then, using those vectors and cosine similarity, it finds the N most relevant sentences for a query (N is a hyperparameter). The other component that takes preprocessed text as input is called the Embedder. The Embedder uses pretrained *word2vec* models, POS tags and the query to create a vectorized representation for each **word**. All the words in the document are concatenated and also the locations of sentence beginnings are stored. So the output of the Embedder is a 2D matrix with dimension $total\_number\_of\_words \times embedding\_size$ and a list of indices of first words for each sentence.

That output is then passed on to the BRC layer that uses Bi-LSTMs and a convolution to represent each sentence. The output of a BRC layer is a matrix with dimension $number\_of\_sentences \times hidden\_size$. The BRC output together with the Retriever output (best sentences indices) is fed into the Sentence retriever that basically takes the BRC representations of the best sentences. Sentence retriever outputs a matrix with dimension $N \times hidden\_size$.

At this point it is important to notice that the output is of a fixed size and that the size is dependent only on hyperparameters. That fixed size is then fed into a usual classifier which can be a multinomial logistic regression, or a deeper model.

4.1. Preprocessing

Our preprocessor consists of five components: alias substitutor, tokenizer, POS tagger, lemmatizer and query expander. We are using competition data from Kaggle.[1] Also, we are doing alias replacement with original gene names because queries contain those names, and because further operations should be easier when we consider all sorts of alias words as one, the same word. First, texts are processed in the alias substitutor part which finds every alias symbol, alias name, previous symbol and previous name of the gene name found in the query for a particular text, and changes them into that name. For the substitution we use data from HUGO Gene Nomenclature Comitee dataset.[2] Tokenizer then tokenizes every sentence from every text. Further, we POS tag every token from texts. The Lemmatizer converts the Penn Treebank tag into a WordNet POS tag, and then uses that tag to perform a much more accurate WordNet lemmatization. In the end, our query expander adds more general elements ("driver" and "passenger") to queries from our loaded data.

4.2. Sentence Retrieval

After preprocessing, the next component is the Retriever. The Retriever uses a *tf-idf* vectorizer fitted on all sentences in the train set. During the training of the vectorizer the terms that appear in less than three sentences are ignored to reduce the size of the vectors.

The query and every sentence in the document are transformed. The most relevant sentences are detected using cosine similarity measure between the query and the sentences. We retrieve the top 30 sentences if the document has more than 30 sentences, otherwise all of them are used.

4.3. Word Embeddings

Word embeddings used are a combination of a pretrained *word2vec* model and additional features. We are using a *word2vec* model that was trained on biomedical literature

[1] https://www.kaggle.com/c/msk-redefining-cancer-treatment/data

[2] https://www.genenames.org/cgi-bin/statistics

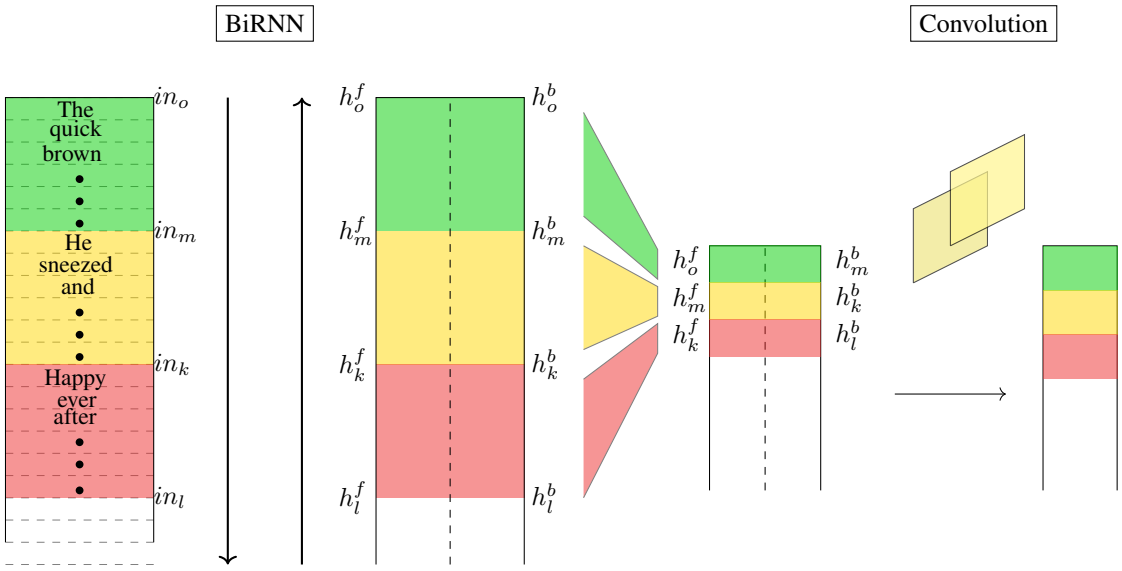


Figure 2: The proposed BRC layer. One color coresponds to the representations of the same sentence. Words of the document are embedded and stacked. Then they are passed through a Bi-LSTM. The relevant embeddings are taken from the Bi-LSTM output and finally passed through a convolution.

from PubMed, which corresponds to our domain of texts.[3] Each word is represented in a 200-dimensional space. Additional features we have added are:

- **Exact query match** - For each token in a query we compare it with our current word and require them to be an exact match. It is a binary vector whose size is the same as query's. Query is of size 4 for each document and it contains: gene, mutation, word "driver" and word "passenger".
- **"Signalling words" match** - We have singled out four words that we find are in relation to our output class ("cancer", "malign", "benign", "pathogenic"). We calculate exact match similarity for each word and encode it as a binary vector of size four. Also, we calculate the similarity between the current word and "signalling word" embeddings as a float giving us a less strict match using the $similarity$ method.[4] It is encoded as a float vector of size four.
- **Negativity match** - Denotes if a given token represents a negative modal verb or a negative pronoun. (e.g. can't, cannot etc.)
- **Token properties** - We use a *Nltk's* POS tagger and encode POS tags as one-hot encoding while all punctuation marks are grouped as one POS tag.[5]

4.4. BRC Layer

The Bidirectional Recurrent Convolutional layer (BRC) is a layer consisting of one BiRNN and a convolution, as can be seen in Figure 2.

The input to a layer, for one example, is created in the following manner. After all the words in all the sentences have been embedded, all of them are stacked one on top of another. That way we have a 2D matrix with the dimensions of *number_words_in_doc x word_embedding_size*.

That input is then passed through a bidirectional LSTM. For each sentence, a forward LSTMSs embedding of the first word in the sentence, and a backward LSTMs embedding of the first word of the following sentence is taken (see figure 2). Those two embeddings contain the context of the text before and after the sentence. Those two embeddings are concatenated and used as a sentence representation. Now we have a 2D matrix of dimensions *number_of_sentences x (2 * lstm_hidden_size* represented by the third column in Figure 2. In that matrix every row is a representation of a sentence. This model of representing sentences is the same as in (Yang Yu, 2016). j The final part is a convolution. The task of a convolution is to add the information of the neighbouring sentence representations to the current sentence representation. The dimension of the filter is *kernel_size x (2 * lstm_hidden_size)* and the dimension of the output is *number_of_sentences x number_of_filters*.

5. Experiments

As mentioned above, we use two test sets. The first one, named stage-1, contains 368 examples with publicly available labels. The second one, named stage-2, contains 986 examples. The labels for the stage-2 are not publicly available and evaluation on the Kaggle webpage is required. For the stage-2 two evaluations are done, public and private.

All the models tested, expect the baseline, follow the previously described structure shown in Figure 1. The results presented as cross entropy losses are shown in Table 4.4..

The baseline uses *tf-idf* measures combined with cosine similarity for retrieval. The best sentences are then represented using the same *if-idf* measures and fed into a multi-

[3] http://evexdb.org/pmresources/vec-space-models/

[4] https://radimrehurek.com/gensim/models/word2vec.html

[5] https://www.nltk.org/book/ch05.html

Model	stage-1	stage-2 private (rank/357)	stage-2 public (rank/357)
ilmirashaim		**2.03(1)**	1.474(304)
Li-Der		2.645(109)	**0.109(1)**
TFIDF_LOGREG(baseline)	1.7	**2.816(166)**	1.645(310)
BRC_LOGREG	1.607		
BRC_CONV	1.029	3.07(227)	**1.207(284)**
BRC_cl_LOGREG	**0.9897**	3.96(317)	1.353(297)
BRC_cl_LOGREG_clipped	1.065	3.946(317)	1.344(297)

nomial logistic regression classifier. Also, the preprocessing is done slightly differently. The aliases are added to the query instead of replacing them in text.

Other approaches displayed in the table are as follows. BRC_LOGREG is a model that uses BRC sentence embedding and a multinomial logistic regression classifier. We see here that despite not optimizing the hyperparameter space (except a bit for the learning rate), and despite not using gradient clipping, our method still outperforms the baseline (TFIDF_LOGREG).

The model named BRC_cl_LOGREG is the same as BRC_LOGREG except that the former uses a convolutional layer instead of just a convolution operation (convolution without the bias and an activation function), as all the other models. As can be seen in Table 4.4. this model performs much better than the BRC_LOGREG model.

When BRCL_cl_LOGREG model is trained with gradients clipped by L2 norm to 1 (BRC_cl_LOGREG_clipped) it performes slightly better or worse depending on the test set.

We have also experimented a bit with one more powerful classifier. The model named BRC_CONV uses a BRC sentence embedding and a classifier containing one convolutional layer with max pooling (because we forgot to remove the pooling and didn't find time to rerun the experiment) whose output is passed on to a fully-connected layer. The convolutional layer has 100 output channels, kernel of height three and width the same as input. This model expectedly outperforms the multinomial logistic regression classi 0 fier models and is currently ranked 284 on the official public Kaggle leaderboard.

In the BRC layer in all of our experiments, the hidden sizes ofhe convolution and both LSTMS are 200 (meaning 200+200 bidirectional LSTM) and 30 best sentences were retrieved in the Retriever component. Each of the 200 filters in the convolution is of the height 3 and width 400 (same as the output of the Bi-LSTM). Training was done using the Adam (Kingma and Ba, 2014) optimizer (SGD for the baseline) with the learning rate of 0.0001 and batch size 20. We evaluate our model on the stage one test set after every epoch and stop training when the loss increases.

6. Problems and Future Work

Unfortunately, a number of ideas have been left unexplored because of the limited time capacity. The full pipeline could be improved using a more powerful classifier, and also a parameter space search (none of the hyperparameters were optimized).

The Retriever component could probably be improved by adding, what could be named, "Gaussian tf-idf retrieval", that would score the saliency of the sentence by the following formula: $s_i = sc_i + 0.7(sc_{(i-1)} + sc_{(i+2)}) + 0.3(sc_{(i-2)} + sc_{(i+2)})$. The idea behind that formula is to retrieve sentences that have small scores but are surrounded by high score sentences.

As to the BRC layer, further experiments with the full convolutional layer instead of a convolutional operation should be conducted. Also, using the exiting hidden states instead of the entering ones (forward LSTMs first word of the following sentence hidden state, backward LSTMs first word of the current sentence hidden state) could improve results.

Also, additional experiments with gradient clipping could be done.

Additional features could be added, such as named entities. Others, such as POS tags, could be improved. The POS tagger we used was not optimized for biomedical texts.

Also, number of sentences we retrieve from each document is fixed (30), and we have plans to optimize that hyperparameter in future.

7. Conclusion

In this paper we have proposed a novel way of representing and processing documents and sentences as fixed-size representations.

We have evaluated our method on the Kaggle task "redefining cancer treatment" and shown that the method displays promising preliminary results that could be further improved through more extensive experiments.

Acknowledgements

We express our sincere gratitude to TakeLab staff for providing us with much needed computational resources.[6]

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

[6] `http://takelab.fer.hr/`

Peter Szolovits Ji Young Lee, Franck Dernoncourt, 2017. *MIT at SemEval-2017 Task 10: Relation Extraction with Convolutional Neural Networks*.

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Krishnadev Oruganty Mahanandeeshwar Gattu Sunil Kumar Sahu, Ashish Anand, 2016. *Relation extraction from clinical texts using domain invariant convolutional neural network*.

Lili Mou Ge Li Yunchuan Chen Yangyang Lu Zhi Jin Yan Xu, Ran Jia, 2016. *Improved Relation Classification by Deep Recurrent Neural Networks with Data Augmentation*.

Kazi Hasan Mo Yu Bing Xiang Bowen Zhou Yang Yu, Wei Zhang, 2016. *End-to-End Answer Chunk Extraction and Ranking for Reading Comprehension*.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344.

How Much Context is Useful in Emoji Prediction?

Hrvoje Bušić, Ante Spajić, Nika Šućurović

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{hrvoje.busic,ante.spajic,nika.sucurovic}@fer.hr

Abstract

Emojis are used in posts on social networks to provide additional information by capturing author's emotion and sentiment. Subjective nature of their selection and complex interaction with the accompanying text results in attempts at making inferences about their appropriate use a hard to solve tasks. In this paper, we tackle the problem of emoji prediction through exploration of the scope of the tweet context that best describes assigned emoji. We train an SVM to extract highly discriminative word n-grams, while also exploring recurrent deep-learning models in an attempt to infer longer text dependencies. Our results have shown that specific subsets of words in tweets are very indicative of the emoji that will be used, reinforcing our belief that brief ideas carry the bulk of the information associated with the emoji.

1. Introduction

Chat apps and social network are used daily by millions of people, who, when expressing themselves use not only words, but also emojis - small digital images used to project an idea or emotion. Emojis have become an integral part of written day-to-day communication, and their use conveys a lot of information about the sentiment of the statement they are used in.

We believe tweets, short message posts on Twitter, are a good representation of informal daily communication where emojis are extensively used - about 20% of the time (Zhao and Zeng, 2016). Tweets are, among other things, used to phrase feelings, opinions and short anecdotes, which makes them favorable for use in interpreting the meaning behind the emojis, and also enables models to learn in what context are certain emojis most often used.

In this paper we decided to tackle the problem of emoji prediction using tweets and framed it as a multi-class classification problem. Given a text of a tweet that previously contained an emoji we build models that predict which emoji was most likely used in the given example. We strive to infer whether the accurate prediction of emojis accompanying tweets depends more on the highly discriminative words and their close neighbors, or on the longer dependencies present in the whole tweet. We use SVMs and a convolutional neural network model to derive answers for the former, and a Bi-LSTM network for the latter question. We also accompanied our findings with a test of significance in order to ensure that our findings are sound and relevant.

2. Related work

Interpretations of different emojis vary vastly across cultures and use-cases. Miller et al. (2016) performed an evaluation in order to prove that asking different people about their perceived underlying meaning of emojis will yield different answers. They asked human annotators to decide the sentiment that emojis evoked, as well as their meaning. It was shown that there were multiple interpretations of the same emojis among annotators and that sometimes the identified meaning was not matching their "original" meaning, designer's intent or even the object they represent.

We base our work and obtain our dataset from Barbieri et al. (2017). The goal of their paper was to classify tweets according to emojis with which they originally appeared. They broke down their original dataset on subsets involving top 5, 10 and 20 most frequently used emojis, where each tweet was always accompanied by only one emoji. They proceed to apply their considered models on each of the subsets and obtain the best results using BiLSTM with pre-trained embeddings. During our research we found significant oversights in the way authors derived the original test sets on which models were evaluated, so we created our own test sets, which we believe make our results more representative, but unfortunately not comparable to the results in the original paper. Moreover, we only consider tweets associated with top 5 and 10 most frequent emojis finding that neither theirs or our model achieve good scores for the top 20, probably due to lack of examples for the less frequent ones. Finally, we have considered a subset of models proposed by the authors and introduced an SVM model as well as convolutional neural network.

In addition to trying to predict the use of emoji we wanted to explore the extent of context that is useful in that task. Can better performance be achieved when taking in consideration whole tweet, or are smaller phrases containing bulk of the information? To see into that we considered a subset of models proposed by (Barbieri et al., 2017) and introduced an SVM model, as well as convolutional neural network, to capture shorter word interactions.

3. Data

We used data provided by Barbieri et al. (2017) originally used for training and testing of their models, in order to compare it to our work and see how performance differs. 40 million tweets were retrieved between months of October 2015 and May 2016, and only those geo-localized in the United States of America are considered. From those tweets, all that contain more than one emoji or an emoji that is not one of the 20 most frequently used in the dataset are discarded and the resulting set contains 589 000 tweets.

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 33–36, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

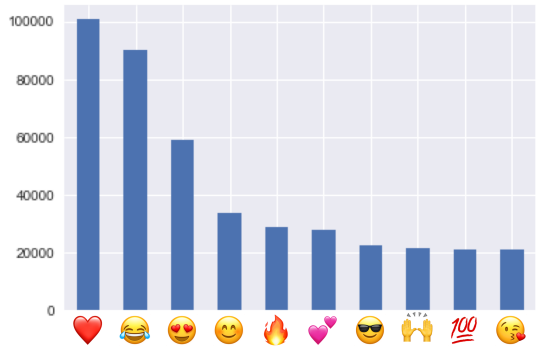


Figure 1: Number of tweets containing each of the top 10 emojis

We noticed that the original training and test sets were not divided properly during their experiment. During their testing, the authors split the 589 000 tweets so that the test set contained close to 4000 examples, while all the rest were used for training. Furthermore, the test set for top 5 most frequently used emojis only contained examples whose assigned label actually belonged to one of 4 emoji types.

After noticing the aforementioned oversights, we proceeded with the creation of 2 new datasets from the original dataset containing top 20 most frequently used emojis. Two datasets are formed out of the top 10 and top 5 most frequently used emojis. The rest of the tweets were discarded. Each dataset is split into a training - 90%, and a testing portion - 10%. We decided to focus on top 5 and top 10 most frequent emojis in our work, since the preliminary experiments we have done on the data involving all 20 emoji classes resulted in very poor classification performance for the least frequent emojis. We assume that the poor performance arises from the small number of tweets being associated with those emojis, so the model tends to go with the class labelings it is confident in and is able to discriminate among better, especially when the context is ambiguous.

In order to compare the performance of our models with those from Barbieri et al. (2017), we also trained and tested our models on the data we have originally been provided with from the authors. Even though we don't find scores in the original paper trustworthy, we provide the comparison in Section 6 for illustrative purposes.

4. Handling tweets

4.1. Tweet preprocessing

Tweets are usually written in an informal way and involve expression characteristic to social networks (e.g. hashtags, mentions of other users and links). Here we describe the techniques that we used to process raw tweets in order to emphasize writing styles and words that denote emotion and sentiment in them, with the goal of improving representations that are provided to our models for training.

Filtering mentions and URLs: Mentions of other users (e.g. @john_doe) are common for tweets, as well as links to other websites. We replace all user mentions with the '<user>' tag and all URLs with the '<url>' tag. In this way, we preserve the information that someone had been addressed and that the tweet referenced another web page respectively, but disregard who exactly was mentioned as well as what the original web page was.

Filtering numbers and hashtags: All numbers in tweets are replaced with the '<number>' tag. Furthermore, hashtags which are used to mark topics and themes with which the author relates the tweet are replaced with the '<hashtag>' tag.

Emphasizing punctuation, repeated characters and 'all caps': Usually, use of punctuation, or more than one punctuation symbol (e.g. 'boss is late again. . . . . '), is very indicative of sentiment, so after that occurs in a tweet we mark it with a '<repeat>' tag. Moreover, excessive repetitions of letters in words also indicate sentiment (e.g. 'realyyyy boring'). Alongside those occurrences, we put an '<elong>' tag (e.g. 'realyyy boring<repeat>'). Lastly, capitalized words are also accompanied by an '<allcaps>' tag in order to point to the strong sentiment expressed.

4.2. Representations of tweets

Here we state the algorithms explored when converting our input tweets to their numerical representations. These representations try to capture the semantic meaning of the tweets while having a constant size, independent of the number of words used in each tweet.

BOW approach: Bag of Words(BOW) is a common approach to numeric text encoding. Given that we have a corpus vocabulary V of finite size, an array of size V with ones in the position of the words present in the text and zeros everywhere else can be created for each of the tweets.

TF-IDF representation: Term Frequency - Inverse Document Frequency(TF-IDF) works by determining the relative frequency of a word in a specific document compared to the inverse proportion of that word over the entire document corpus. Words that are common to a small group of documents are more discriminative and significant than words common to the whole corpus, and therefore have a higher TF-IDF value. While it can be used as a standalone representation, this metric can also be used to weight the Bag of Words representation.

Word embeddings representation: Word Embeddings(WE) is a word representation that maps words to high dimensional vectors of real numbers. This representation tries to capture the multiple contexts in which a word can appear in a text and therefore guarantees that words that appear in similar context will have vectors that are very close to each other.

The advantage of WE compared to the other word representations is that each word is represented by a fixed-size vector regardless of the used vocabulary. In tweet classification problem, given a fixed window of size V, tweets can be represented by a VxD matrix, where D is the dimension of word embedding vectors.

In our experiments, we used the pre-trained set of Word Embeddings for tweets, GloVe (Pennington et al., 2014). These vectors were trained on 2 billion tweets containing 1.2 million words.

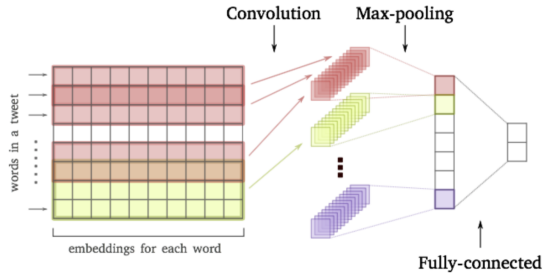


Figure 2: Design of Convolutional Neural Network used for classification of tweets.

5. Models

We employed several models that use different classification algorithms: Logistic Regression, Support Vector Machines, Convolutional Neural Networks, and Bidirectional Long Short-Term Memory Networks. These are briefly explained below.

5.1. Logistic Regression

Logistic regression is a statistical model used when the dependent variable is binary or dichotomous, while independent factors of variables can be categorical or numerical variables. Binary logistic regression model can be generalized to more than two levels of the dependent variable, resulting in categorical outputs.

For our baseline model, we used such Logistic Regression classifier using previously explained Bag of Words vector for representation of tweets as input. We went for the most common configuration using a One-vs-Rest classification scheme with default parameters provided by sklearn [1] implementation. When training the model we iterated 1000 times for the solvers to converge.

5.2. Support Vector Machines

SVMs are performant classification algorithms that work well with high-dimensional data and whose aim is to separate samples from different classes with as wide of a margin as possible. They are applicable both as linear binary classifiers and non-linear complements after kernel trick. In our experiments, we utilized linear SVM classifier for the purpose of exploring the importance certain keywords and phrases have on the accuracy of prediction using n-grams. The input for our model were tweets represented as TF-IDF weighted BOW vectors. We also used word embeddings in tweet representations, but had gotten slightly worse results.

Utilizing 5 way K-fold on the train set we tried out values for penalty parameter $c \in [0.1, 2.0]$ and n-gram sizes ranging from $(1, 1)$ to $(1, 6)$, ending up with $c = 1$ and $(1, 4)$ n-grams as optimal choices.

5.3. Convolutional Neural Networks

Having a prominent place in computer vision, CNNs have successfully been applied to NLP tasks as well, including sentence classification (Kim, 2014). For CNN to be applied to the textual data, be trained in mini-batches and able to understand the contextual relationship between numerical representations of words, each tweet is assumed to be consisting of a maximum n_{max} words and is represented by a matrix of size $n_{max} \times D$, D being the dimensionality of word embeddings. In case of absence of the needed number of words, the matrix is zero-padded. Figure 2 depicts the application of CNN on text: multiple filters of different window size (pictured in different colors) are convolved over the tweet matrix. Max-pooling is applied to the produced layer to decrease the number of data points. Finally, softmax layer serves as a classifier.

In our model inputs were parsed tweets turned to embeddings using GloVe. Embedding were formed by concatenating 40 word embeddings sequentially, one word below the other) and then convolving on top of them. These original embeddings were additionally trained by our algorithm, thus achieving a better score than without it. We designed our CNN to contain 15 filters spanning across three, four and five words (5 filters for each length). Max-pooling is done for each of those 15 filters and than softmax layer is applied to get the classification. We experimented adding additional fully connected layers of different sizes after the convolutional layer but that resulted in worse performance.

We trained our model using mini-batches of 512 tweets and Adam optimizer (Kingma and Ba, 2014). As we did not have a separate validation set, we picked 10% of our training set to be used as one. Training was done until we saw a drop in performance on the set used for validation.

5.4. Bidirectional Long Short-Term Memory Networks

Recurrent neural networks are the current state-of-the-art algorithms for sequential data analysis. Capable of preserving internal memory, RNN's can remember important features from the input and use it in predicting what's coming next. LSTMs are built on top of the ideas behind recurrent neural networks and are capable of learning long-term dependencies. The units of an LSTM are used as building units for the layers of an RNN. This extended memory can be seen as a gated cell that decides whether or not to store or delete information based on the importance it assigns to it. Bidirectional LSTMs train two LSTMs on the input sequence: the first one on the input sequence as-is and the second one on a reversed copy of the input sequence. This provides the additional context to the network and we wanted to use that in order to capture longer text dependencies.

The input format of tweets presented to Bi-LSTM is the same as the one used with CNN, and the embeddings were again additionally trained by our algorithm. This approach yielded better results than one that kept embeddings fixed. The model was trained in the same way as the CNN: mini-batches of size 512, using Adam optimizer and a validation set extracted as 10% of the training set.

We used the architecture proposed by Barbieri et al. (2017), setting the size of a single-layer Bi-LSTM model to 128. Moreover, we experimented with the addition of new layers to try and extract deeper understanding of the text, and explored different layer sizes, but failed to improve the

[1] http://scikit-learn.org/stable/index.html

Table 1: Results

	Top 5			Top 10		
Model	Precision	Recall	F1	Precision	Recall	F1
Baseline	0.59	0.60	0.58	0.44	0.46	0.42
SVM	0.61	0.62	**0.61**	0.46	0.48	**0.46**
CNN	0.59	0.60	0.59	0.44	0.47	0.43
Bi-LSTM	0.59	0.60	0.58	0.43	0.46	0.42

Table 2: F1 scores on the original dataset.

Model	Barbieri et al.	This paper
Top 5		
Baseline	0.58	0.60
Best performer	0.63	0.64
Top 10		
Baseline	0.41	0.46
Best performer	0.47	0.47

F1 score.

6. Results

On the newly filtered datasets, our SVM model turned out to be the best performer on both getting the highest F1 score but also achieving highest precision and recall for the top 3 emojis, showing that those three can easily be discriminated among themselves and others via indicative keywords and phrases. Seeing how the model works best emojis with the most examples and noticing the considerable variation in frequencies among different emojis (Figure 1), we created two additional datasets in which we randomly selected the same number of tweets belonging to each class, but did not achieve significant improvement.

The CNN model achieved second best performance on all sets; outperforming the baseline and the Bi-LSTM model, which did very poorly, not even exceeding the baseline F1 score. Bad scores of the Bi-LSTM indicates that longer text dependencies are not as important in the prediction of emojis.

In Table 2 we present the result of our models alongside those of Barbieri et al. on their original dataset, for the comparison. Both our baselines were simple Logistic Regression models that just represented tweets differently. Their best performer was a char-BiLSTM while our was SVM as mentioned before. On our own datasets, we achieve slightly worse results shown in Table 1, but trust that those are more representative.

Our best performer for this task was the SVM model so we tested it against our baseline to show that their results significatly differ with the 2,5% level of significance. To achieve this we used approximate randomization test (Yeh, 2000) with 1000 iterations, coming to a conclusion that our improvement is statistically significant with a p-value of less then 0.01.

7. Conclusion

Throughout our paper we explore the question of how information connected to the used emoji is contained in a tweet, and demonstrate that highly selective words and phrases in tweets are most indicative of that. Through exploration of different models, we show that simpler to understand and train models, such as SVM, can achieve better performance than more complex ones, like the considered CNN.

Future work should consider whether expanding context with which current models work (e.g. by including information from links to which tweets point to or conversations current tweets are a part of), or the addition of more data can yield better performance.

References

Francesco Barbieri, Miguel Ballesteros, and Horacio Saggion. 2017. Are emojis predictable? *Proceeding of the European Chapter of the Association for Computational Linguistics (EACL)*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980*.

Hannah Miller, Jacob Thebault-Spieker, Shuo Chang, Isaac Johnson, Loren Terveen, and Brent Hecht. 2016. Blissfully happy" or "ready to fight": Varying interpretations of emoji. *Tenth International AAAI Conference on Web and Social Media (ICWSM 2016)*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Alexander Yeh. 2000. More accurate tests for the statistical significance of result differences. *In Proceedings of COLING 2000*.

Luda Zhao and Connie Zeng. 2016. Using neural networks to predict emoji usage from twitter data. Stanford University.

The Analysis of Preprocessing Methods for the Purpose of Detecting SMS Spam

Frano Ćaleta, Ivan Rezić, Damjan Vučina

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{frano.caleta,ivan.rezic,damjan.vucina}@fer.hr

Abstract

According to a recent study, there are more than two billion mobile devices in the world. The massive growth of the number of mobile phone users in the last decade has lead to the fact that more than ten trillion messages are sent between users each year. Such immense numbers have a negative effect on the world of messaging as well. Many surveys indicate that most mobile phone users have experienced some sort of SMS spamming and are more likely to change their operator since they perceive mobile marketing messages as a sort of spam. The everlasting fight with SMS spamming is indeed challenging, mostly because messages are often quite short and more importantly, written using informal language. Thereby, common message filters' performance deteriorates rapidly. Having such shortcomings in mind, this paper focuses on providing an extensive analysis of the supplemented SMS spam collection and offers comprehensive comparison between a handful of renowned machine learning models.

1. Introduction

Short Messaging Service i.e., *SMS*, has in recent years become widely used and extremely affordable telecommunication service package. A substantial increase in the number of phone users paired with the existence of such easily accessible and used platform, consequently made this to become a multi-billion dollar industry.

As the messaging platform grew in popularity, so did the number of unsolicited commercial advertisements sent to mobile phones using text messages. Moreover, in recent years SMS pricing has dropped significantly. In China, for example, a single SMS costs less than a thousandth of a dollar which resulted in approximately every third SMS being identified as a spam. Furthermore, some carriers themselves have been spamming their clients with various marketing techniques via SMS platform. This resulted with mobile phone users becoming increasingly concerned regarding their personal freedom being intruded. Message spamming is a practice that has been around for quite some time, but initially started taking advantage of e-mail clients. Nowadays, it has become even more of a problem, since in some countries spam messages contribute to the cost for the receiver. Moreover, this results in unnecessary bandwidth usage and raid of personal privacy.

The major downside of text messages from the perspective of user is that they are often quite short and written using informal language which makes it hard for the renowned e-mail filters to be easily adapted for the new platform. Due to the short length of the messages, the number of features that can be used for their classification is rather small. Another issue is that databases used for SMS spam filtering are quite small when compared to the datasets used in detecting e-mail spam. The problem these flawed and incomplete tools are facing is the legitimate possibility of filtering out possibly significant messages.

Having these shortcomings in mind, this paper provides an extensive analysis of this problem and applies different popular machine learning models comparing their effect. The goal is to further explore the previously described problem from a number of different perspectives applying various machine learning models and providing the reader with the best available tools for spam filtering.

Section 3 presents the dataset we have used for research purposes. Section 4 describes our preprocessing techniques. Section 5 and Section 6 demonstrate the implementation steps and the results.

2. Related Work

Spam messages in mobile communication platforms have been troubling us for years. Various approaches to the idea of filtering unwanted content have been found, published in literature and proposed as revolutionary tools to fight this stubborn enemy of the modern age.

Some papers have a rather general proposal of blacklisting the sender of the message identified as a spam (Delany et al., 2012). This approach simply stores the sender's number in a special list so any subsequent messages from this number will be discarded. This proposition, however, often falls short since the sender can rather easily change his number and the number we collected quickly becomes obsolete.

Previous authors attempted to provide a baseline spam detecting system by testing a variety of renowned models and deducing the best approach (Almeida et al., 2011). The authors of this paper came to a conclusion that linear SVM offers the best performance outperforming other classifiers even though most approaches filtered out only about half the spam messages.

The same authors in a different paper attempted to ensure that the SMS Spam Collection building process is not negatively affected by the reusage of the same message sources (J. M. G. Hidalgo and Yamakami, 2012). They concluded that merging collections that share sources does not necessarily lead to a drop in the performance.

3. The SMS Spam Collection

The core part of any scientific research is the acquired dataset that is to be analyzed. The common shortcoming that publicly available datasets are often identified with is the absence of representative data.

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 37–40, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

SMS spam datasets are by no means different than the ones mentioned above. A few SMS spam databases have been collected and made available, but genuine spam messages still remain hard to identify. Having these shortcomings in mind, for the purposes of this paper we decided to use the dataset built by (Almeida et al., 2011). This dataset has been compiled from several renowned sources and to this day remains the largest spam dataset available.

The corpus includes the dataset that has been manually extracted from UK forum. The data available on this forum is quite suitable for our needs since this web page emerged from people's anger with spamming and resulted in many spam messages being uploaded to this site.

The corpus also includes a subset of random non-spam SMS messages collected at the University of Singapore from volunteers who were made aware that their contributions will be made publicly available. This dataset is available at following URL.[1]

Another sources for the dataset are the SMS samples publicly available within the PhD available at [2] and a corpus[3] that has already been used by (Cormack et al., 2007).

This amounts to 4827 genuine messages and 747 spam messages, i.e., a total of 5574 messages and makes the dataset we have used the largest available SMS spam corpus that currently exists.

4. Preprocessing Data

In this section we present different preprocessing methods and the libraries we used to implement them.
Each line in the SMS spam collection dataset represents the message itself and its corresponding class (spam or ham). Figure 1 shows the structure of spam and ham messages, and as expected, ham messages are usually shorter than spam messages. According to a previous study on data duplicates, the presence of duplicates in our sample may lead to a loss in classification accuracy(Kołcz et al., 2003), so we decided to train our models on the original dataset, and on the one where duplicates are dropped. Other than that, we also decided to replace the digits with character 'N' to preserve phone numbers' structure (J. M. G. Hidalgo and Yamakami, 2012).

4.1. Vectorization

Since our data consists of text, it must be encoded as either numbers or vectors prior to training our models.
We used two different vectorizers provided by sckit-learn (Pedregosa et al., 2011), *TfIdfVectorizer* and *CountVectorizer*. Both vectorizers use the "Bag of words" model, which means that they assign a weight to each word only based on its frequency in the dataset. TF-IDF stands for "term frequency-inverse document frequency", the weight assigned to each word (token) depends on its frequency in a document but also how frequent that word is in the rest of the dataset. On the other hand, *CountVectorizer* just counts the frequency of a word and assigns that count to word's weight. We performed several language-specific preprocessing methods. According to previous authors, stemming in general may hurt model's accuracy (Almeida et al., 2011). Nevertheless, it doesn't really state which stemming algorithms tend to hurt the accuracy so we decided to use *Porter Stemmer* since it produces the best results as compared to other stemmers (Jivani, 2011). Also we used *Word Net Lemmatizer* because lemmatization is able to identify similar and unnecessary words which could lead to improvement in detecting SMS spam (Jivani, 2011). Furthermore, we decided to use n-gram lexical features (unigrams and bigrams) and stop words removal.

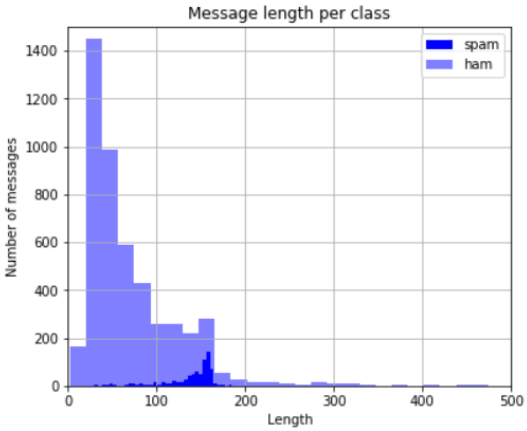


Figure 1: Text message length per class

5. Implementation

In the previous section, we proposed multiple preprocessing methods. In this section, we exclude some of these methods from our final evaluation.
So far we have decided to train our models on four different datasets shown below:

- d1: Dataset including duplicates
- d2: Dataset excluding duplicates
- d3: Dataset including duplicates; digits are replaced with letter 'N'
- d4: Dataset excluding duplicates; digits are replaced with letter 'N'

Furthermore, we decided to implement other language specific preprocessing methods:

- tok1: N-grams
- tok2: Stop words removal
- tok3: Porter Stemmer
- tok4: Word Net Lemmatizer
- tok5: Every word is individual token

Additionally we implemented two different vectorizers for each of these methods. To avoid extensive search for

[1]http://www.comp.nus.edu.sg/rpnlpir/downloads/corpora/smsCorpus/
[2]http://etheses.bham.ac.uk/253/1/Tagg09PhD.pdf
[3]http://www.esp.uem.es/jmgomez/smsspamcorpus

Table 1: Accuracy and f-1 score for each classifier using preprocessing methods

Tok	Vectorizer	SVM acc / f1	MNB acc / f1	RFC acc / f1
Tok1	CountVectorizer	0,98 / 0,91	0,98 / 0.92	0,96 / 0,85
Tok2	CountVectorizer	0,98 / 0,91	0,98 / 0,92	0,96 / 0,86
Tok3	CountVectorizer	0,98 / 0,91	0,98 / 0,91	0,98 / 0,85
Tok4	CountVectorizer	0,98 / 0,92	0,98 / 0,92	0,97 / 0,85
Tok5	CountVectorizer	0,98 / 0,93	**0,9865 / 0.94**	0,97 / 0,85
Tok1	TfidfVectorizer	0,98 / 0,93	0,98 / 0.92	0,96 / 0,82
Tok2	TfidfVectorizer	0,98 / 0,92	0,98 / 0,91	0,97 / 0,82
Tok3	TfidfVectorizer	**0,9857 / 0,93**	0,98 / 0,92	0,97 / 0,81
Tok4	TfidfVectorizer	0,98 / 0,92	0,98 / 0,92	0,97 / 0,84
Tok5	TfidfVectorizer	0,98 / 0,92	0,98 / 0.92	0,96 / 0,82

Table 2: Final experiment and evaluation results

Tok	Dataset	Model	Vectorizer	Hyperparameters	acc/f1
tok3	d4	SVM	TfidfVectorizer	C=10, kernel=linear	**0.9874 / 0.94**
tok3, tok1	d4	SVM	TfidfVectorizer	C=10, kernel=linear	**0.9874 / 0.94**
tok3	d3	SVM	TfidfVectorizer	C=10, kernel=rbf, gamma=0.1	**0.9874 / 0.94**
tok3, tok1	d3	SVM	TfidfVectorizer	C=10, kernel=rbf, gamma=0.1	**0.9874 / 0.94**
tok5	d1	MNB	CountVectorizer	alpha=1	0.9865 / 0.94
tok3	d1	SVM	TfidfVectorizer	C=10, kernel=rbf, gamma=0.1	0.9857 / 0.93
tok1	d1	SVM	TfidfVectorizer	C=10, kernel=sigmoid, gamma=1	0.9848 / 0.93
tok5	d2	MNB	CountVectorizer	alpha=0.05	0.9848 / 0.93
tok5	d4	MNB	CountVectorizer	alpha=0.05	0.9839 / 0.93
tok5	d3	MNB	CountVectorizer	alpha=0.05	0.9839 / 0.93
tok4	d1	SVM	TfidfVectorizer	C=1, kernel=linear	0.9839 / 0.93
tok3	d2	SVM	TfidfVectorizer	C=10, kernel=sigmoid, gamma=0.1	0.9830 / 0.92
tok3, tok1	d2	SVM	TfidfVectorizer	C=10, kernel=sigmoid, gamma=0.1	0.9830 / 0.92

each possible combination of previous methods, we decided to train our models on non-edited data (this includes duplicates and digits are not replaced with letter 'N') to exclude some of these methods in case they cause a drop in accuracy or f1-score.

5.1. Choosing the Best Preprocessing Methods and Models

We trained three different classifiers :

- *Support Vector Machine* (SVM)
- *Multinomial Naive Bayes* (MNB)
- *Random Forest Classifier* (RFC)

We used SVM that has already been used by previous authors (Almeida et al., 2011), however, none of them applied any preprocessing methods. The reason we used MNB is because according to *sklearn* documentation, it works well with word counts, which is basically what CountVectorizer does (Pedregosa et al., 2011). Finally, we implemented RFC because it was widely used on Kaggle, a platform for predictive modelling in which users compete to produce the best models for predicting and describing various datasets. [4] The idea is as following:

1. We load the initial dataset without any editing
2. We split the dataset into train and test set by randomly choosing 80% as training and 20% as test set.
3. For each preprocessing method, vectorizer and model combination, we have performed grid search on train set with 5 fold cross validation also know as k-folding. The main reason to use k-folding is that the k-fold cross-validation estimator has a lower variance than a single hold-out set estimator, which can be very important if the amount of data available is limited (like our dataset). If you have a single hold out set, where 80% of data are used for training and 20% used for testing, the test set is very small, so there will be a lot of variation in the performance estimate for different samples of data, or for different partitions of the data to form training and test sets. K-fold validation reduces this variance by averaging over k different partitions, so the performance estimate is less sensitive to the partitioning of the data.
4. After extracting optimal hyperparameters for our preprocessing method and model combination we have evaluated each of them with accuracy score and other main classification metrics, such as recall, precision

[4] https://www.kaggle.com/

and f1-score. We were searching for a model with highest accuracy and f1-score while having precision greater than 0.98. Reason for choosing precision as an important metric is because we do not want to miss out non-spam messages as they can be very important to text message users.

Table 1 shows the vectorizer, preprocessing method, accuracy and f-1 score for each model we used. First of all, we may conclude that Random Forest Classifier performs poorly compared to the other two models, the accuracy rate is similar but the f1 score is far lower so we will not be using it in our further experiments. Secondly, the best results were given by CountVectorizer combined with Multinomial Naive Bayes, additionally, it is obvious that this combination has some performance drops using any language specific preprocessing method. Furthermore, it is fair to say that generally Support Vector Machine performs better using TfidfVectorizer than CountVectorizer, and surprisingly, using Porter Stemmer produces the best results.

6. Final Experiment and Evaluation

After finding the best preprocessing methods and models on our unedited dataset we wanted to implement them on different previously mentioned types of datasets (d1, d2, d3, d4). In this section, we used these methods (and their combinations) on other three datasets. First dataset is the inital dataset without duplicates, and the other two are the ones where numbers are replaced with letter 'N', with and without duplicates. Once again, we perform the steps mentioned in the previous section, but this time for TfidfVectorizer we only use SVM while applying the best preprocessing methods (tok1, tok2 and tok3). Additionally, for CountVectorizer we only use Multinomial Naive Bayes without any language specific preprocessing method (tok 5) because the previous experiment has shown that these combinations produce the best results. For more information, look up Table 2. We can see that first four models have same results, as some other rows with worse accuracy and f1. We can conclude that some tokenizations are unnecessary as they make no difference while increasing model complexity. So in that sense, we will take TfidfVectorizer + linear SVM (row 1) as our best model. In Table 3, we can see how our model compares to (Almeida et al., 2011) models in accuracy and in Table 4 we can compare it to (Dima Suleiman, 2017) model's f1 score. It is important to note that this comparison is not exactly fair because we didn't use identical test sets in both cases as author did.

Table 3: Our model (bolded) in comparison to models from (Almeida et al., 2011).

Model	Accuracy score%
SVM, TF-IDF, Stemming	**98.74**
SVM	97.64
Boosted NB	97.50
Boosted C4.5	97.50
PART	97.50
MDL	96.26

Table 4: Our model (bolded) in comparison to models from (Dima Suleiman, 2017).

Model	F1 score
Random forest	0.97
SVM, TF-IDF, Stemming	**0.94**
Deep Learning	0.87
Naive Bayes	0.78

7. Conclusion

The task of automatic SMS spam filtering is a difficult challenge. While the low number of features is one of the problems, the main problem is the lack of large public datasets. In this paper, we have reviewed various techniques for text processing and provided the reader with comprehensive results. We have performed multiple preprocessing methods and vectorizer-model combinations while providing optimal hyperparameters through grid search. Furthermore, we demonstrated that usually stigmatized stemming combined with SVM and TfidfVectorizer yields best results, while some other tokenization combinations turned out to be unnecessary. For future work, we suggest further TF-IDF + Porter Stemmer exploring and improving Random Forest Classifier since it produced the highest f-1 score in previous work (Dima Suleiman, 2017).

References

Tiago A. Almeida, José María G. Hidalgo, and Akebo Yamakami. 2011. Contributions to the study of sms spam filtering: New collection and results. In *Proceedings of the 11th ACM Symposium on Document Engineering*, DocEng '11, pages 259–262, New York, NY, USA. ACM.

Gordon V. Cormack, José María Gómez Hidalgo, and Enrique Puertas Sánz. 2007. Feature engineering for mobile (sms) spam filtering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07.

Sarah Jane Delany, Mark Buckley, and Derek Greene. 2012. Sms spam filtering. *Expert Syst. Appl.*, 39(10), August.

Ghazi Al-Naymata Dima Suleiman. 2017. Sms spam detection using h2o framework, 11.

T. A. Almeida J. M. G. Hidalgo and A. Yamakami. 2012. On the validity of a new sms spam collection.

Ms. Anjali Ganesh Jivani. 2011. A comparative study of stemming algorithms.

Aleksander Kołcz, Abdur Chowdhury, and Joshua Alspector. 2003. Data duplication: an imbalance problem?

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Plain Text Enrichment Using Tweets And Emojis

Marin Drabić, Dora Marković, Luka Suman

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{marin.drabic,dora.markovic,luka.suman}@fer.hr

Abstract

Emoji usage as a part of social media text messages has dramatically increased in the past decade. Emoji, a small digital image or icon used to express an idea or emotion, combined with textual content creates a form of modern way of communication. Emojis come in various shapes (facial expressions, common objects, places, animals etc.) and sometimes the same emoji even has a different meaning, due to different context and viewer's interpretation. The purpose of this paper is to create a model which predicts the placement and type of emoji in text-based tweet messages. We are presenting a two-step solution: first we are predicting the location of emojis in the tweets and second, after predicting emoji locations, we are trying to determine the type of emoji in a specific location. We propose a solution to both problems (location and type) using the same models: one baseline (MFC), two linear models (SVM and AdaBoost) and one deep learning model (BLSTM).

1. Introduction

In the past few years new popular way of communication, which combines textual and visual content, has standardized. This, so-called emoji concept has created some complications for standard information systems, which now did not just have to deal with text, but also images. Emoji became part of everyday communication, and are available on many online platforms, such as Facebook, Twitter, Instagram, Whatsapp, Snapchat etc. Emojis are ideographic; meaning that they represent ideas or concepts that are independent of a specific human language. So, naturally, the question arises: how can an appropriate emoji be predicted in a sentence?

Emoji, as a part of the textual conversation, captures really well face-to-face conversation. That is why people use words such as hahaha, lol, omg, etc. Lately, emojis are often used as a way to express emotion while communicating online. Also, emojis are a very good indicator of sarcasm usage in conversation. We would like to enrich plain text with emotion indicators (emoji) in hopes of better relaying the writer's thoughts. Reading a message such as "*I love you.*" has its meaning, but adding a heart-shaped emoji at the end gives the message a bigger impact.

Emoji prediction research presents a relatively new task. Emojis are used on daily basis and often represent the replacement for some basic words. However, often misleading and ambiguous, emojis can easily create additional challenges for information systems. In other words, not only does the semantics of the text have to be handled but also the semantics of the images. But is it possible to find a way for replacing emoji with a text content for easier text manipulation or to use textual information as an input for emoji prediction?

In this paper, we are proposing a way for the use of textual information to predict emojis associated to text. We collect textual data from Twitter and process it. This is done in two steps. First, a prediction whether the emoji should be placed at a certain position is made. This is a binary prediction. Second, the type of the emoji is predicted as a multi-class problem.

2. Related Work

Emoji semantics and usage have been studied with distributional semantics, with models trained on Twitter data (Barbieri et al., 2017) and has shown that textual information can be used to predict emojis associated to text. In the mentioned research, they train models based on Long Short-Term Memory networks (LSTMs), assuming that computational models can understand the better semantics of emojis. They propose an idea of artificial intelligence system for emoji prediction as a part of a better understanding of natural language tasks (generating emoji-enriched social media content, enhance emotion/sentiment analysis systems etc). They employ a state-of-the-art classification framework to automatically predict the most likely emoji a Twitter message evokes. Their model is based on Bidirectional Long Short-term Memory Networks (BLSTMs) with word representations and character-based representation of tokens. For word representation, generated word embeddings are learned together with the updates to the model. The character-based approach learns representations for words that are orthographically similar, and for that, alternatives of the same word type occurring on (in this case) Twitter should be easily handled. Their task is to predict the single emoji that appears in the input tweet regardless of its position.

3. Two-step Solution

In our paper, we are also dealing with the emoji prediction for Twitter messages, but as an upgrade to the previous paper (Barbieri et al., 2017), we are proposing a model able to predict more than one emoji per tweet. Furthermore, our model is able to predict the position of emoji in the given tweet and to choose the appropriate type of emoji from a list of available emojis (Figure 1). So, our task is done as a two-step solution.

First, we are doing a prediction of positions where to place the emojis. This is handled as a binary classification; emoji will either be placed in some position in the tweet or it will not be. Second, after predicting the exact positions for emojis, for every position that emoji is determined

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 41–45, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

to be, appropriate emoji is chosen from a list of available emoticons (Figure 1). Both solutions use same models: one baseline, two linear models and one deep learning model.

4. Dataset

We used the dataset from competition Semeval 2018 Task 2 – Multilingual Emoji Prediction (Barbieri et al., 2018) which includes tweets that contain only the 20 most frequent emojis which are presented in Figure 1. The dataset consists of 500k tweets in English which were retrieved with the Twitter APIs, from October 2015 to February 2017, and geolocalized in the United States. Due to Twitter restrictions tweets are not allowed to be shared directly, so we had to download tweets using Twitter Crawler. We were able to collect only 473.622 tweets as JSON objects because of unavailability errors.

For the process of extracting and cleaning tweets, we used Emoji Extractor provided by Semeval 2018 competition. We modified it so it suits our needs for collecting a list of texts without emojis and list of labels separately for emoji locations and emoji types. All hyperlinks are removed from the tweets, user initials are changed to "@user", consecutive punctuations are reduced and we lowercased all textual content to reduce noise and sparsity. For experimenting, we created one corpus with punctuations and hashtags and another corpus with punctuations but without hashtags. Our intuition is that training on corpus with punctuations but without hashtags would give the best results. That is because hashtags greatly increase the size of vocabulary and bring no importance for generalization (for example, "*better luck next year juniors* 😉*#powderpuffchamps #eatorbeeaten #ididntouchtheballonce*"). This assumption showed to be true in our experiments.

We agreed to use tweets that only have one or multiple emoji types in tweets but that are not followed one by another because for every location just one emoji type is taken as a true label. For example, "*Buckwheat chocolate chip pancake. As healthy as it gets!*😊😍*#pancakes #breakfast*" is not acceptable because for the same location there are two separate emojis. But "*Girls night* ❤️❤️❤️*@ Saint Andrews* 😍" is acceptable because it can be transformed to "*Girls night* ❤️*@ Saint Andrews* 😍". We also agreed on taking 30 words per tweets since only a fraction of them are longer than that.

For corpus without hashtags we got 473.459 good tweets and for corpus with hashtags 473.370 tweets that we used as final input tweet text to our models. With hashtags, there are lesser samples because the size of some tweets increased to more than 30 words per tweet. The vocabulary size for corpus without hashtags contains 137k different words and with hashtags increases to 288k which is more than double. Labels for emoji locations are taken for every location as 1 or 0 whether an emoji was there in the original tweet. Emoji type labels are mapped to a number between 0 and 20 where 0 marks no appearance of emoji and the rest describes 20 most frequent emojis. Both labels are set to size 31, taking also the location before the first word in tweet into consideration, so that all labels are uniform in size. Examples are shown in Table 1.

Figure 1: The 20 most frequent emojis that we use in our experiments

Table 1: Example of input tweet text, emoji location and emoji type labels for original tweet with removal and no removal of hashtags

Went to Sleep in LA 😎, Woke Up in Vegas!!!! #Life ❤️	
Without hashtags	
Input	went to sleep in la , woke up in vegas !
Location	00000100000100000000000000000000
Type	00000700000100000000000000000000
With hashtags	
Input	went to sleep in la , woke up in vegas ! #life
Location	00000100000010000000000000000000
Type	00000700000010000000000000000000

5. Models

5.1. MFC Baseline

Most of the labels in our input data are equal to zero. This is because we use zero padding when tweets are shorter than 30 words and because tweets usually have only a few emojis in them. This causes a class imbalance problem where the zero class outnumbers other classes 15 to 1. For this reason, we used a stratified majority class classifier (MFC). This baseline gives high accuracy and low recall. We are later able to perform a sanity check by comparing all of our models to this classifier.

5.2. SVM

We used the LinearSVC implementation from scikit-learn (Pedregosa et al., 2011). It is based on the liblinear library for large-scale linear classification. It is by far the fastest model to train since it is highly optimized for large datasets. For inputs to LinearSVC, TF-IDF is computed on the collection of tweets. Then, for every position between words, a new example is generated: a 2*k array containing the k left and k right TF-IDF values of words. In our experiments, we set k to 3. Additionally, for every generated example, emoji location label and emoji type label are taken for that particular position between words. Hyperparameter C is fine-tuned using testing. For emoji location prediction the best obtained is 2^7 and for emoji type prediction 2^6. SVM takes a few minutes to train.

5.3. AdaBoost

AdaBoost is an ensemble which uses weak classifiers. A total of 100 decision tree classifiers of depth 2 are used. Input examples and labels are computed the same as for LinearSVC using TF-IDF. Both the number of trees and their depth were optimized using grid search on 5 folds and a fixed test set (10%). It performs better than the SVM, but

it takes much longer to train. Average training time was about 30 minutes.

5.4. BLSTM

As our final model, we used a Bidirectional Long Short-Term Memory recurrent neural network. BLSTMs are capable of modeling sequential and temporal aspects of data (Goodfellow et al., 2016). By using a second LSTM layer which reads from the end, a BLSTM can take into account both halves of the input when predicting the output label. We used Keras and its implementations of the various required layers. Keras (Chollet et al., 2015) is a high-level API for neural networks which can be run on top of TensorFlow. TensorFlow (Abadi et al., 2015) is an open source software library for high-performance numerical computation. TensorFlow enables computation on GPUs which is much faster than on CPUs. For our training, we used the following hardware: Intel i5-6600K CPU and Nvidia 1070Ti GPU.

Firstly, the input examples need to be created. A tokenizer is used to map every word in a tweet to a unique integer. This makes it easier to handle string data. Then every tweet shorter than 30 tokens (including words and interpunction) is padded with zeros at the end. Padding is required by Keras. Model's task will be to label words with either 0 or 1 depending on whether an emoji should be placed after the word. An additional zero is added at the beginning so that the BLSTM can predict an emoji at the start of the tweet. Input labels (which are binary for locations) are one-hot encoded. This removes ordering and is also required by Keras.

The first layer is the embedding layer which serves as an input layer. It contains a 2D matrix of dimensions $V \times E$ where V is the vocabulary size and E is the embedding size. Each word from the vocabulary has a one-dimensional vector associated with it. These vectors are stored in rows and are accessed through the indexes from the tokenizer. Keras uses mini-batches with 32 examples per batch. This means that the input is a two-dimensional array ($B \times L$) where B is the batch size and L is the length of every example. The output gains an additional dimension ($B \times L \times E$).

We tried training our own embeddings from scratch, but it proved to be too difficult because tweets often contain misspelled words and slang. Luckily, pretrained embeddings for Twitter exist. We used GloVe embeddings (Pennington et al., 2014) trained on 2 billion tweets. GloVe embeddings have a vocabulary of 2.7 million words and they are available in 4 different vector sizes: 25, 50, 100 and 200. Embedding size is a hyperparameter and 200 proved to be the best since it gives the most information. We decided to fine tune the pretrained embeddings in order to increase model performance. Although this greatly increases the number of parameters, the resulting gain in performance is worth it.

The second layer is a bidirectional layer which is composed of two LSTM layers. First LSTM layer reads the input from the start and the second LSTM layer reads the input from the end. Outputs from these two LSTM layers are concatenated and sent to the next layer. We tried using the average of the two LSTM layer outputs, but this resulted in worse performance. Both LSTM layers have the same number of hidden units. A number of hidden units represents a hyperparameter. Higher hidden size increases the expressive power of the model, but it comes at a cost in computation times. The number of hidden units was determined through testing.

The last layer is a dense layer using the softmax activation function. Training the model is done using the Adam optimizer with cross-entropy loss. We also used dropout between the embedding layer and the BLSTM layer. A dropout rate of 50% is used in all experiments. Blocking half of the inputs during training helps prevent the model from overfitting. The number of epochs was controlled by using early stopping during training. Loss on the validation set is monitored and training is stopped when the loss did not improve for two consecutive epochs (this is also called patience). Most experiments lasted between 4 and 12 epochs with each epoch taking 15-20 minutes on our GPU.

6. Results

6.1. Location Prediction

Table 2 shows the location prediction results. All of the models beat the baseline in recall and F1 score. Both linear models perform worse than the BLSTM model in all metrics except precision. BLSTM proved better than AdaBoost through a permutation test. Table 5 shows the results of the BLSTM model selection. Increasing the embedding size leads to a general increase in F1 score. Increasing the number of hidden units in the BLSTM layer lowers the precision and increases the recall on the test set.

Table 2: Emoji location prediction results

Model	Accuracy	Precison	Recall	F1
Baseline	0.8491	0.0831	0.0840	0.0836
SVM	0.5936	0.7419	0.1362	0.2302
AdaBoost	0.6919	0.7164	0.1707	0.2758
BLSTM	**0.8707**	**0.7805**	**0.1735**	**0.2839**

6.2. Emoji Type Prediction

Predicting emoji types is much harder than just predicting their locations. Results in Table 3 show a great decrease in F1 score for all models when compared to the results in the previous table. SVM did not manage to get a better F1 score than the baseline.

Table 3: Emoji type prediction results

Model	Accuracy	Precison	Recall	F1
Baseline	0.9534	0.0125	0.0125	0.0125
SVM	**0.9560**	0.0215	0.0109	0.0115
AdaBoost	0.9289	0.0779	0.0186	0.0175
BLSTM	0.9392	**0.1958**	**0.02389**	**0.0329**

Table 4: Example of BLSTM prediction

Tweet	Yesss that's right my girl made court!! Love her ❤@ Southeastern University
BLSTM input	yesss that's right my girl made court ! love her @ southeastern university
Location prediction	0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Emoji type prediction	0 0 0 0 4 14 14 4 4 14 14 14 14 14 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Final prediction	0 0 0 0 0 0 0 0 4 14 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Result	Yesss that's right my girl made court!! 💕Love 💜her 💜@ Southeastern University
Tweet	Thanks to @boomburritos for another reason to look forward to Friday and to doodle at my desk ❤
BLSTM input	thanks to @user for another reason to look forward to friday and to doodle at my desk
Location prediction	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Emoji type prediction	0 0 10 0 10 10 0 10 10 0 10 10 0 15 15 15 10 10 10 10 10 10 10 0 0 0 0 0 0 0 0
Final prediction	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 10 0 0 0 0 0 0 0 0 0 0 0 0 0
Result	Thanks to @boomburritos for another reason to look forward to Friday and to doodle at my 😘desk 😘
Tweet	When you got a friend who is literally the cutest elf you've ever seen 💕..
BLSTM input	when you got a friend who is literally the cutest elf you've ever seen ..
Location prediction	0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Emoji type prediction	8 0 0 0 8 0 0 3 8 2 3 3 3 2 2 2 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Final prediction	0 0 0 0 0 0 0 0 0 2 0 3 3 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Result	When you got a friend who is literally the 😍cutest elf ✨you've ✨ever 😍seen ..
Tweet	Last night birthday party #family 😊@ 809 Bar & Grill
BLSTM input	last night birthday party @ 809 bar & grill
Location prediction	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Emoji type prediction	8 8 8 8 11 11 19 19 19 19 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Final prediction	0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Result	Last night birthday party ✨#family @ 809 Bar & Grill

Table 5: BLSTM model selection results

Embedding size = 200				
Hidden units	Accuracy	Precison	Recall	F1
200	0.8642	0.7982	0.1687	0.2785
500	0.8665	0.7895	0.1700	0.2798
1000	0.8829	0.7135	0.1787	0.2859
Embedding size = 100				
Hidden units	Accuracy	Precison	Recall	F1
200	0.8659	0.8132	0.1727	0.2849
500	0.8685	0.7738	0.1700	0.2787
1000	0.8869	0.6878	0.1801	0.2854

Table 4 shows examples of outputs using BLSTM. We can see that the model predicts series of locations in order to roughly guess where the emoji actually is. Emoji types are much harder to predict and here our system's predictions are similar to the target emojis. For example, the heart emoji ❤is often replaced by similar emojis such as 💕and 💜.

7. Conclusion

Emoji, as a part of a modern way communication, may help the user, providing the way of expressing not always possible by words, but also can create a confusion due to its ambiguity and possible misunderstanding due to viewer's interpretation. In this paper, we observe the use of emoji and suggest a way of predicting a position and type of emoji used in tweets. By exploring relations between word and emoji, we are trying to predict placement and the most probable type of emoji for text-based tweet message type of data.

The problem proved to be very difficult. Reading emotions from the grammatically correct text is hard in itself. Even humans would have a hard time predicting emojis in tweets. Learning a model from tweets is hard mainly because they are unstructured and very diverse in the words they contain. Nevertheless, our model gives meaningful results.

As future work, the model could be improved by a bigger dataset with some form of tweet preprocessing. BLSTM model could be expanded with more layers and optimizations. Lastly, the final prediction could have more than two steps for location and emoji type prediction.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vin-

cent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Francesco Barbieri, Miguel Ballesteros, and Horacio Saggion. 2017. Are emojis predictable? *CoRR*, abs/1702.07285.

Francesco Barbieri, Jose Camacho-Collados, Francesco Ronzano, Luis Espinosa-Anke, Miguel Ballesteros, Valerio Basile, Viviana Patti, and Horacio Saggion. 2018. Semeval-2018 task 2: Multilingual emoji prediction. In *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, LA, United States. Association for Computational Linguistics.

François Chollet, Joseph J. Allaire, Yuan Tang, Daniel Falbel, Wouter Van Der Bijl, and Martin Studer. 2015. Keras.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Distinguishing Genetic Mutations for Tumor Growth with NLP

Filip Floreani, Matija Haničar

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
filip.floreani@fer.hr, matija.hanicar@fer.hr

Abstract

Cancer tumors can have thousands of genetic mutations, and various mutations have a different effect on tumor growth. The problem is to determine which variations contribute to cancer growth and which do not. Currently, this task is done manually by an expert. This method can be error-prone because it relies entirely on expert knowledge. In this paper, we present a technique which utilizes some basic NLP approaches for the task of mutation classification.

1. Introduction

Personalized medicine helps doctors learn about a person's genetic makeup and how their tumor grows. With this information, doctors hope to find prevention, screening, and treatment strategies that may be more effective than just regular broad-spectrum treatment. They also want to find treatments that cause fewer side effects than the standard options. By performing genetic tests on the cancer cells and healthy cells, doctors may be able to customize treatment to each patient's needs. The diagnostic reports of these tests, as well as other clinical literature, is consulted by researchers and oncologists when they need to decide if a given gene mutation has something to do with tumor growth or not. If a particular mutation is said to affect tumor growth, then it is labeled as a driver; otherwise, it is called a passenger.

The main problem with this hands-on approach is the overwhelming number of scientific papers, diagnostic reports and other text-based clinical sources that have been written about genes and their various mutations (also called variations). These texts are incredibly technical and complex to comprehend, so it takes a lot of time to find the right text with the right information about a given gene-variation combination. Solving this issue would be a big step in the direction of personalized medicine, which is precisely what this paper is trying to help accomplish.

We demonstrate a simple NLP approach to analyzing textual clinical data and classifying gene-variation combinations accordingly. Our system extracts and selects the most important features from the corpus, but it also makes room for some specialized genetic features that drastically improve the final classification score. Additionally, if new genetic features are discovered in the future, our approach makes it easy to incorporate them into the pipeline.

In this paper, we first start by describing the initial data set and the way it is transformed to include as much useful information as possible (sections 3 & 4). By using a simple feature generation/selection model, we build a set of features for the classification process (section 5). The results are analyzed by comparing calculated metrics to existing models and by doing ablation studies on the included genetic features, which in turn reveals their importance for successfully solving the task at hand.

2. Related Work

Even though we could not find other papers to reference ourselves on, we will be referencing our model on a few existing solutions found on the competition website and through various article posts online. Some of the existing work overlaps with our approach, mostly in the preprocessing section while generating feature weights. Regarding classification, we have seen multiple different paths. Aly Osama (Osama, 2017) used a Doc2Vec feature generator and used the full gene and variation names as additional features for a dense neural network. Jorge Muñez (Muñez, 2017) built multiple deep learning models, as well as regular ML models and used only the first couple of thousand words for classification. On the other hand, we have Bhuvaneshwaran's (K, 2017) simplistic approach by using a TF-IDF vectorizer in a pipeline with a linear SVM, which gave unexpected results. All but a select few reported using only features extracted from the clinical texts, without trying to use the rest of the data to create more information for the classification process. In this paper, we take an approach that combines most of the tried and tested features from earlier attempts and add additional features that go a bit out of the realm of regular NLP, but show promising results.

3. Data Set

The dataset used for training and evaluating the model is composed of 3320 manually labeled medical reports and is freely available on Kaggle.[1] Actual data is pulled from the OncoKB knowledge base, which contains information about the effects and treatment implications of specific cancer gene alterations.[2] Every gene-variation pair has an associated clinical report or some other type of text source. The pair can be assigned one of nine available classes which are not labeled in the original data set on Kaggle, but have been discovered by examining OncoKB and are as follows:

1. Likely Loss-of-function
2. Likely Gain-of-function
3. Neutral

[1] https://www.kaggle.com/c/msk-redefining-cancer-treatment
[2] http://oncokb.org

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 46–49, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

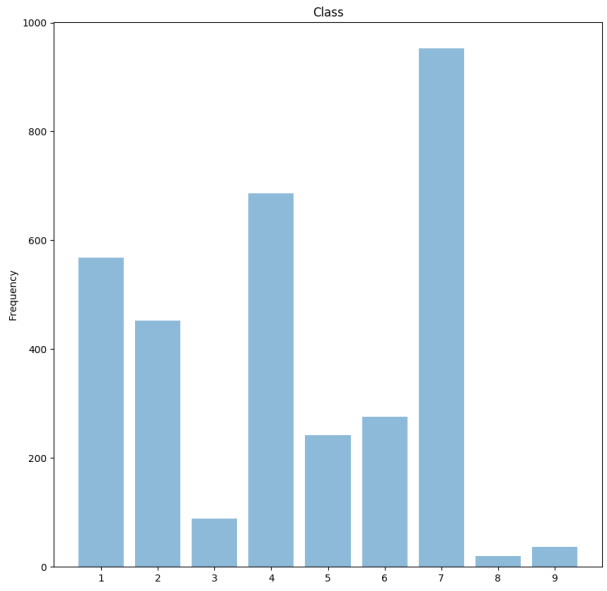


Figure 1: Distribution of data samples per class.

4. Loss-of-function
5. Likely Neutral
6. Inconclusive
7. Gain-of-function
8. Likely Switch-of-function
9. Switch-of-function

4. Data Formatting

The distribution of data samples across classes is shown in Figure 1. A quick look makes it easy to see that, as expected, the samples are not equally distributed. In fact, there is a gross distance between some classes, which obliges us to do some oversampling before we do anything else with the data. It is important to note that we do this only on the training data to obtain a better result. There should be no over or undersampling done on the test data, as it mirrors real-world samples. Now, as we are handling medical data, and it being a bit unorthodox to oversample by interpolation of existing samples, we opted for naive random oversampling for classes that are well underrepresented, and undersampling for those that are overrepresented. This is done by calculating the mean between the number of samples of the most frequent and the most infrequent class and then sampling all other classes to that number.

It is worth mentioning that we have experimented with SMOTE (Synthetic Minority Over-sampling Technique) by first calculating weights from the clinical text and then applying oversampling, but have not achieved satisfactory results, and so have abandoned that direction.

5. Model Overview

Our model consists of two main components:

1. A preprocessor used for feature generation and selection by using a combination of a TF-IDF vectorizer followed by a χ^2-test to select the best feature weights for the next step.
2. A postprocessor for managing a set of optimized classifiers, and also for calculating & plotting score metrics based on predicted class labels.

Even though the model is simple and somewhat crude, the intention was to use the open text as a basis for feature extraction and combine those features with more complex genetic features to obtain a quality set that covers both the language processing and the clinical aspect of the problem.

5.1. Preprocessor

As previously mentioned, the core of the used dataset is the extensive collection of unstructured clinical text. As there are no apparent features to extract from it, we decided to create a feature generator based on two steps: TF-IDF vectorization and the Pearson's χ^2-test.

The TF-IDF vectorization process is preceded by word tokenization and the removal of stop words. Besides the standard set of English stop words, taken from the NLTK's corpus module, we also decided to remove words that are commonly associated with scientific papers (e.g., ' et al.', 'author', 'figure', 'table') and are objectively irrelevant for later classification.[3] We have also decided to use n-grams between one and three words in length, as using longer n-grams did not show a significant change in the final results but has extended processing time by a fair amount.

Running the TF-IDF over the whole corpus gave us some 250 thousand features, so it was clear that we needed to filter out the best ones from that set. This has been done by applying a χ^2-test over the feature set and selecting the top ten percent as the free text features that will later be used in the postprocessing stage. Table 1 shows the top three best features extracted per class.

Regarding the more complex genetic features, we used a modified toolkit for protein feature engineering developed by Ofer et al. (2015). For genetic variations that have a format `char_digits_char`, with the first and last character being from the 'amino acid alphabet' (a list of amino acids coded by 22 single characters), we used the Grantham score as an additional feature. Grantham score attempts to predict the distance between two amino acids, in an evolutionary sense, and its result is a number between 5 and 215 (Grantham, 1974). For those variations that do not conform to the format mentioned above, we calculate the mean of the Grantham scores for the whole data set and assign the mean to those variations. As all the features we extracted previously are in range 0 to 1, we decided to scale this feature down to the same range.

One additional genetic feature we tried to extract and use in the classification was the location of the variation in the gene, designated by a 1 to 7 digit number in the variation name, but have decided against using it in the final model, since the classification score did not significantly improve.

[3] www.nltk.org

Table 1: Most predictive features per class.

Class	Feature names
1	'growth' 'conformation' 'uveal melanoma'
2	'lymphoma' 'mcf10a' 'mcf10a cells'
3	'hmg' 'tsc patients' 'nonsense'
4	'r175h' 'mutant p53' 'eecs'
5	'pathogenicity' 'sf9' 'protein'
6	'i157t' 'lynch syndrome' 'bach1-m299i'
7	'mpm' 'stat5' 'breakpoint'
8	'd-2hg' 'chondrocytes' 'idh2'
9	'y641' 'ctcf' 'cic–dux4'

5.2. Postprocessor

Once the preprocessor has successfully extracted useful features from the data, we can begin to feed that information to our classifier set. Our first baseline is a low one, a simple dummy classifier tuned to classify every sample to the most frequent class. For a more realistic baseline, we used a multinomial Naive Bayes classifier. We measured them up to some more complex models, specifically random forest, gradient boosting, and linear SVM classifiers.

The dataset was split into training, test and a validation set (70: 20: 10 ratio). First, to fine tune the hyperparameters, a grid search mechanism was implemented for every classifier in the set (except the dummy classifier, of course). The search was run over the validation set so that the hyperparameters would not be both trained and tested on the same data, which would end up overfitting the final model and giving out incorrect results.

Having the best hyperparameters we can obtain, we fit each classifier over the training set and then calculate the predictions on the test set. Both the training and the test set have manually assigned labels so that we can calculate the classification score for the predicted classes.

We ended up calculating the accuracy, precision, recall and F_1 score for each of the classifiers. As we have previously sampled our classes to the same number of samples, we decided to calculate the macro-average for all scores, apart from the accuracy.

6. Results and Analysis

Table 2 contains the scores of the predictions given by the classifier set. The random forest & the gradient boosting classifiers offer comparable performance for the given data set and are significantly better compared to our baseline. It is worth noting, though, that the linear SVM classifier is not that far off from the baseline, which by itself has an interestingly high score. Referencing other work found on the competition website, we determine that our model does not lack behind by performance scores.

6.1. Ablation Study

We did an ablation study on the genetic amino acid distance feature by removing it from the model and then re-training and re-testing it, the results of which are shown in Table 3. A significant drop can be noticed in the metrics of most of the used classifiers, almost 50 percent in some of them, indicating that this feature is crucial for our model. This also notes the importance of complex genetic features in general for the task at hand, making a right heading for future research.

Table 2: Weighted precision, accuracy, recall and F_1 score. The upper half consists of previously reported results, the lower half are our models.

Model	A	P	R	F1
Osama (2017.)	0.560			
K (2017.)	0.647			
Multinomial NB	0.564	0.515	0.523	0.519
Linear SVM	0.615	0.558	0.568	0.563
Random Forest	**0.678**	**0.669**	**0.742**	**0.704**
Gradient Boost	0.657	0.650	0.732	0.689

Table 3: Amino acid distance feature ablation study results

Classifier	P	R	F1
Linear SVM (with feature)	0.558	0.568	0.563
Linear SVM (without feature)	0.285	0.297	0.291
Random Forest (with feature)	0.669	0.742	0.704
Random Forest (without feature)	0.318	0.364	0.339
Gradient Boost (with feature)	0.650	0.732	0.689
Gradient Boost (without feature)	0.358	0.412	0.383

7. Conclusion and Future Work

Personalized cancer treatment is a complex area of oncology and genetics, as it requires combing through excessive amounts of textual oncologic & pathologic data to find the answer to a growing tumor. We formatted this problem as a combination of a fundamental feature extraction problem and a complex genetic feature generation task. We used common NLP feature extraction techniques to generate useful features from textual clinical data, and have also used the information packed inside the genetic variation labels to create additional features that had a substantial impact on the final results. By combining this information, we have shown that exploiting both the free text and the genetic data can be beneficial for this type of tasks. Our model was measurable to some of the best scoring models in the competition, but it also managed to keep the general concept concise and straightforward. Further work would be aimed at creating improved classification models, possibly some deep learning ones, to try and further increase the classification score and reduce the processing time. We

would also be examining additional complex genetic features that could be incorporated into the existing pipeline to boost performance.

Acknowledgements

We would like to wholeheartedly thank Assoc. Prof. Aaron Quinlan and his team at Quinlan Lab at the University of Utah for their work on the mapping of the Grantham amino acid matrix.

References

R. Grantham. 1974. Amino acid difference formula to help explain protein evolution. *Science*, 185(4154):862–864.

Bhuvaneshwaran K. 2017. Redefining cancer treatment - linear svc. Blog article. Available online (https://www.kaggle.com/bhuvaneshwaran/redefining-cancer-treatment-linear-svc).

Jorge Muñez. 2017. Personalized medicine: Redefining cancer treatment with deep learning. Blog article. Available online (https://medium.com/clusterone/personalized-medicine-redefining-cancer-treatment-with-deep-learning-f6c64a366fff).

Aly Osama. 2017. Doc2vec with keras. Blog article. Available online (https://www.kaggle.com/alyosama/doc2vec-with-keras-0-77).

Do You Need To Worry About Post History?

Juraj Fulir, Ivan Mršić, Tea Đuran

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{juraj.fulir, ivan.mrsic, tea.duran}@fer.hr

Abstract

With the rising numbers of people reporting diagnoses of mental disorders, along with the number of social media users, there has been a growing number of studies which report a strong connection between language use and mental health. In this paper, we approached the problem of detecting depression from the language used on Reddit posts by using six different models and observing the results when analysing isolated posts opposed to analysing the whole post history. Surprisingly, our results indicate that examining the whole post history does not give more insight into mental health state than examining a single post.

1. Introduction

Depression is one of most common mental disorders, affecting more than 350 million people of all ages worldwide. With the average age of the onset of depression being only 19, it can be detrimental to a person's life as it can affect a lot of choices in the age where people would normally be at their most productive. While it is not always possible to notice the early signs of a developing mental health issue, it is important to try to detect it as early as possible in order to react and intervene in time. While there are over 200 classified forms of mental illness, this paper will primarily focus on depression and its connection to the language people use on social media.

Due to people having access to various (anonymous) platforms on which they can express their feelings, share life events and participate in various online communities, social media was bound to become an everyday part of peoples lives. Given the fact that there is an ever growing amount of linguistic data which people release to the online world every time they post something, it is no wonder that there has been a rising number of studies which attempt to leverage the language people use to make conclusions about their mental health.

We follow multiple such pieces of research which claim that there are certain words and linguistic patterns which can give an insight into a persons mind and personality. Some of the examples of language characteristic for people who suffer from depression are excessive use of the first-person pronoun, negations, and exclamations. The results of these studies were used in the construction of the features and are explained in more detail in Section 4.1.

In this paper, we present four shallow and two deep learning models which we used for detecting depression from Reddit posts. For each model we tried to model the subject in two ways: looking at his writings post by post, and looking at all of his history up until given point in time. Our main focus is the comparison of these two approaches and evaluating whether the one holds performs better than the other.

After evaluating all of the models discussed in Section 4. and performing permutation tests we concluded that the whole post history does not provide more information about mental health than isolated posts from the user. In the end, we take our results with a grain of salt, as we do acknowledge that some deep learning models could make more use from the entire history of users posts being provided.

2. Related work

The entry point for our project was the paper (Losada and Crestani, 2016), which described the process of constructing the dataset with the intention to study the difference in language use between people who are diagnosed with depression and those who were not, along with analyzing the language progression through time. The authors also defined their evaluation metric, Early Risk Detection Error (ERDE), which will be used to evaluate models defined in this paper and is explained in Section 3.1.

In (Benton et al., 2017) the authors used deep learning in the prediction of mental health, which has inspired us to include two deep learning models in our work as their system resulted in significant gains in predicting mental health with limited data.

Following those two fundamental papers, there were several others which tried to determine the exact language used by people with different mental disorders. In (Gkotsis et al., 2016), the authors extracted data from the 10 most relevant subreddits such as /r/depression, /r/bipolar, etc. and analyzed the language used in them in order to determine linguistic features.

The authors of (R Brubaker et al., 2012) observed the language of post-mortem comments left on the online profiles of deceased people in order to detect the language a person uses when in emotional distress. The result of their work was a *codebook* of rules which were used to classify posts as emotional distress. They concluded that linguistic style can be a strong marker for expression of emotional distress.

In (De Choudhury et al., 2013), the authors measured depression using data from Twitter. They approached the problem from several angles, studying the engagement on the social media through the number and frequency of posts, amount of *links* shared, a number of *question-centric* posts, and a measure called *insomnia index* which quantified the pattern of posting to see if the user is more active during the night. Due to the Twitter social interac-

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 50–54, University of Zagreb,
Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Table 1: Examples of positive and negative subjects

	Positive	Negative
Date	2015-06-09 06:40:45	2015-07-15 15:34:01
Text	*I found this to be pretty good information. My main issue is just trying to feel like I'm worth doing anything positive for.*	*These people trying to make a case for coal by saying that the wind never blows again once it stops and the sun never rises again once it sets, and you want me to not judge them for their stupidity?*

tion structure, the authors were able to construct social graphs for users using the information about retweeted posts and aimed to measure whether their graph was egocentric. Finally, they constructed the *Depression lexicon* using the data available in the "Mental health" category of Yahoo! Answers. We were able to utilize their work and make the most out of the lexicon when designing the features for our models.

Similar to them, (Coppersmith et al., 2015) used data collected from Twitter for one of their previous works, (Coppersmith et al., 2014), in order to explore the language used by people suffering from various mental disorders, from ADHD to SAD. The same Twitter dataset was used by (Resnik et al., 2015), who used supervised LDA to analyze depression-related language on Twitter.

3. Dataset

The dataset used in this paper comes from (Losada and Crestani, 2016). It consists of posts submitted on Reddit, from both people who once explicitly stated they were diagnosed with depression and other users, some of which may suffer from depression. As described in (Losada and Crestani, 2016), the data was collected from Reddit because of its terms and conditions which allowed the use for research purposes, contrary to data which could be collected from Facebook and Tumblr. Each subject consists of a series of posts containing the anonymized ID of the user, title (if it is available), date of posting, information about the post and the text body of the post. The training set consists of total 486 subjects, 83 positive and 403 negatives. An example from both positive and negative subject can be found in Table 1. The dataset statistics are shown in the Table 2. Not all posts contain a title, as they are likely a comment and not the top post. Likewise, not all posts contain text as they are probably a post asking a question or providing a URL. These characteristics were used during feature design when calculating post to reply ratios.

Some preprocessing steps needed to be done before using the data. We have performed tokenization followed by part-of-speech tagging which was later used in designing features and finally, lemmatization in order to reduce the noise which came from informal writing on Reddit.

Table 2: Dataset statistics

Feature	**P**	**N**	**Total**
Num of subjects	137	755	892
Num of writings	49580	481873	531453
Avg posts per subject	361.9	638.2	595.8

3.1. Early risk detection error

Several metrics were used to evaluate the models. An evaluation metric proposed by (Losada and Crestani, 2016) called ERDE (*early risk detection error*), an official measure of the competition, was used along with the standard classification measures such as precision, recall, accuracy, and F-measure. ERDE was defined as follows: "An early risk evaluation metric needs to take into account the correctness of the (binary) decision and the delay taken by the system to make the decision. The delay is measured here by counting the number (k) of distinct textual items seen before giving the answer. " The formal representation can be found below.

$$ERDE_o(d,\ k)\ =$$

$$\begin{cases} c_{fp} & d\ =\ P\ \&\ ground\,truth\ =\ N\ (FP) \\ c_{fn} & d\ =\ N\ \&\ ground\,truth\ =\ P\ (FN) \\ lc_0(k)\cdot c_{tp} & d\ =\ P\ \&\ ground\,truth\ =\ N\ (TP) \\ 0 & d\ =N\ \&\ ground\,truth\ =\ N\ (TN) \end{cases}$$

The function $lc_0(k)$ is a cost function which grows with k and penalizes the late discovery of a positive subject.

4. Models

4.1. Features

Our features can be divided into three larger groups: expression features, stylistic features, and lexicon features. We performed feature ablation among groups, and it turned out that model works best with all of the features turned on.

Expression features are supposed to model the subjects general word usage and keep track of specific habits.

- Emoji: we scanned each subject for emojis and calculated the ratio of emojis to overall words used in the post/reply.
- Tf-idf: we calculated the tf-idf for post/reply. We used unigrams and bigrams, with the post and reply vectors being 300 and 75 long. We settled on vector lengths and n-grams used based on the initial experiments.
- Url: binary feature indicating whether the post/reply contains URL.

Stylistic features model the user's style of writing and his Reddit activity.

- Exclamation: after our initial dataset analysis, we came to the conclusion that non-depressed subjects use exclamation in the larger amount than depressed ones.

- Negation: following related work we used the evidence of negation as an indicator of a depressed subject.
- Post ratio: the ratio of subjects posts and his overall activity. Overall activity is defined as a number of both posts and replies the subject made.
- Pronouns: in a similar fashion to negations, we calculated the pronoun ratio indicator.
- I: one of the most prevalent themes in the related work was the subject's usage of the pronoun I when communicating.
- Reply ratio: the ratio of subjects replies and his overall activity.
- Verb tenses: research has indicated that depressed subjects use the past tense more often than non-depressed ones. We encode each tense as a separate vector, defined as a ratio of that tense among all tokens.

Lexicon features were features which calculated the ratio of a certain lexicon in comparison to the overall number of words in post/reply. We used the curse, disclosure, relationships, symptoms, and treatment lexicons.

4.2. Shallow learning models

We used the linear SVM, Gaussian Naive Bayes, Logistic Regression, and Random Forrest Classifier as the shallow classifier models. For both the SVM and the Logistic Regression performed hyperparameter search over value of C $2^{(i)}$, $i \in \{-10, 10\}$ ($2^{(-10)}, 2^{(-9)}, ..., 2^{(8)}, 2^{(9)}, 2^{(10)}$). As for the Random Forrest we performed the grid search over the values of *n_estimators* and *max_features*. When calculating ERDE post by post we were looking at most the last 10 subjects posts and if more than half of them could be classified as the post of a depressed subject the subject was classified as such. A grid search was performed with the 5 fold cross-validation.

4.3. Deep learning models

We used two fully connected architectures as deep classifiers. Both models use ReLU as activation for hidden layers and a sigmoid for the output layer. For both architectures we performed hyperparameter grid search over values of the learning rate $\eta \in \{10^{-3}, 5 * 10^{-4}, 10^{-4}\}$ and the regularization coefficient $\lambda \in \{10^{-2}, 10^{-3}, 10^{-4}\}$. Both models were trained using the Adam optimizer with exponential decay of learning rate by $\delta = 0.999$ per step. Models were trained for 100 epochs and the best model was chosen based on the validation result on the test set.

The first deep model uses a siamese architecture with three separated input layers (Figure 1): one for title embedding, one for content embedding and one for the feature vector. Those layers are mapped with a single fully-connected layer each. Those mappings are then concatenated and passed to the rest of the fully-connected classifier. By doing this we hope to automatically capture whether an input is a post or comment and process each independently of the other. Additionally, this architecture provides a content independent feature selection and composition from the feature vector.

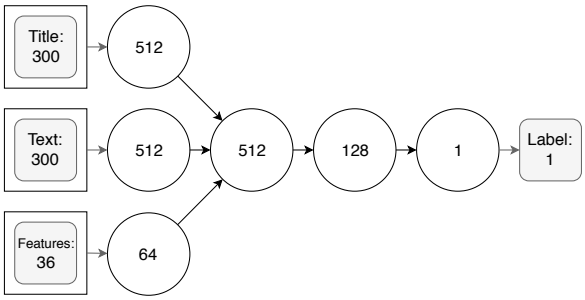


Figure 1: Siamese architecture: inputs are mapped separately and then concatenated for the rest of the classifier.

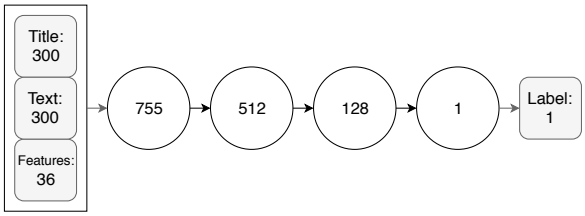


Figure 2: Standard architecture: inputs are concatenated and mapped jointly for the rest of the classifier.

The second deep model uses a standard fully-connected architecture that treats the title embedding, content embedding and the feature vector as a single concatenated input (Figure 2). This architecture uses a number of parameters comparable to the first deep model. With this architecture we want to check if there was any performance benefit from separating the input embeddings and features or does the model perform better if left to perform its own feature selection.

5. Results

Somewhat unintuitively the best ERDE results were achieved by the Random Forrest Classifier which seemed to have performed the worst on the recall of positive subjects. Our explanation for this discrepancy is that other classifiers classified too many negative examples as positive. Results for the post by post models are shown in the Table 3 and for the history models in the Table 4.

We calculate ERDE metric for the post by post approach in the following way: given at most 10 latest data points, we calculate the positive/negative ratio. If the ratio exceeds 0.5, we define the subject as depressed. For historical approach, we simply waited until the model predicts the subject as positive and then concluded its classification label as such. The precision, recall, and F1 scores are calculated in the same way for both history and post by post model.

Same as the dataset authors we showcase our accuracy, precision, recall, F1 score, and two ERDE measures. They differ by the threshold applied, one has a 5 post threshold and the other 50 post.

After acquiring the ERDE results for all of the modes, we performed the paired permutation test over the results to see whether the difference between the approaches is statistically significant. Each time the test was performed on 10000 permutations. The percentiles are shown in the Table 5. As

can be seen from it, not only there is not a statistically significant difference between models, the percentiles can be around 0.5. Even though results seem to discourage usage of the historical approach one could argue that models which we employed do not have high enough capabilities to model posts in such a way, while the LSTM and convolutional models could take more advantage of post context.

6. Conclusions and future work

Our results show that there should not be much of a difference in results no matter whether we use the post by post or historical approach. In the light of those results, we would argue the usage of the post by post approach due to it being less hardware intensive. Our results are encouraging as well, as our best classifier has beaten the baseline which dataset authors have provided. Our comparison of deep architectures shows that separating the input vectors and processing them separately does not significantly affect the overall performance. Another thing we noticed was the somewhat random scoring when talking about the ERDE metric; we believe that modeling it to take into the account the actual post time along with posts number would be more helpful. Our reasoning for this lies in the fact that dataset contains quite a huge discrepancy in average post numbers and times between posts.

In future work, we would like to experiment with the LSTM and convolutional deep models, which should be able to model the subject's psyche as the context from his posts. Their historical approach might benefit more from subject's previous posts since not every post or comment might be representative of the subject's overall mental state. From our results of comparison between deep architectures, it might be preferable to use the siamese architecture for context modeling architectures. This way the model can learn to recognize the different contexts that might exist between posts and comments, which may be used for a more accurate prediction. Furthermore having a huge corpus of Reddit posts would be helpful, as we could model the Reddit word embeddings which would fit our task. In the end, we believe that expanding the dataset with information about subreddits and full comment chains since there is some evidence that communication with other people can be indicated when it comes to detecting depression and other mental disorders.

References

Adrian Benton, Margaret Mitchell, and Dirk Hovy. 2017. Multitask learning for mental health conditions with limited social media data. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 152–162.

Glen Coppersmith, Mark Dredze, and Craig Harman. 2014. Quantifying mental health signals in twitter. In *Proceedings of the Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*, pages 51–60.

Glen Coppersmith, Mark Dredze, Craig Harman, and Kristy Hollingshead. 2015. From adhd to sad: Analyzing the language of mental health on twitter through self-reported diagnoses. In *CLPsych@HLT-NAACL*.

Munmun De Choudhury, Michael Gamon, Scott Counts, and Eric Horvitz. 2013. Predicting depression via social media. *ICWSM*, 13:1–10.

George Gkotsis, Anika Oellrich, Tim Hubbard, Richard Dobson, Maria Liakata, Sumithra Velupillai, and Rina Dutta. 2016. The language of mental health problems in social media. In *Proceedings of the Third Workshop on Computational Lingusitics and Clinical Psychology*, pages 63–73.

D. Losada and F. Crestani. 2016. A test collection for research on depression and language use. In *Proc. of Experimental IR Meets Multilinguality, Multimodality, and Interaction, 7th International Conference of the CLEF Association, CLEF 2016*, pages 28–39, Evora, Portugal, September.

Jed R Brubaker, Funda Kivran-Swaine, Lee Taber, and Gillian Hayes. 2012. Grief-stricken in a crowd: The language of bereavement and distress in social media. 01.

Philip Resnik, William Armstrong, Leonardo Claudino, Thang Nguyen, Viet-An Nguyen, and Jordan Boyd-Graber. 2015. Beyond lda: exploring supervised topic modeling for depression-related language in twitter. In *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*, pages 99–107.

Table 3: Post by post model results

Model	Accuracy	Precision	Recall	F1	ERDE5	ERDE50
Linear SVM	0.73	0.90	0.73	0.79	**0.073**	0.070
Gaussian Naive Bayes	0.45	0.89	0.45	0.56	0.083	0.101
Logistic Regression	**0.92**	0.88	**0.92**	0.89	0.126	0.117
Random Forrest Classifier	0.91	0.88	0.91	0.89	0.074	**0.055**
Siamese FC	0.77	0.87	0.77	0.82	0.18	0.16
Standard FC	0.80	0.87	0.80	0.83	0.19	0.16

Table 4: History model results

Model	Accuracy	Precision	Recall	F1	ERD5E	ERDE50
Linear SVM	0.84	0.87	0.84	0.86	0.102	0.090
Gaussian Naive Bayes	0.89	0.86	0.89	0.87	0.093	0.108
Logistic Regression	**0.92**	0.86	**0.92**	0.88	0.108	0.075
Random Forrest Classifier	0.84	0.87	0.84	0.85	0.124	0.124
Siamese FC	**0.92**	0.91	**0.92**	**0.92**	0.56	0.51
Standard FC	**0.92**	0.91	0.91	**0.92**	0.53	0.48

Table 5: Permutation test percentiles

Model	ERD5E	ERDE50
Linear SVM	0.912	0.495
Gaussian Naive Bayes	0.197	0.289
Logistic Regression	0.468	0.669
Random Forrest Classifier	0.874	0.664
Siamese FC	0.370	0.228
Standard FC	0.722	0.491

Ensemble Learning for Emotion Intensity Problem in Tweets

Robert Injac, Dominik Stanojević

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{robert.injac,dominik.stanojevic}@fer.hr

Abstract

Ensemble learning methods are often used to achieve the most efficient models. In this work, we use three different models as solutions to the task of predicting emotion intensity in tweets. We use sentiment and emotion lexicons along with word embeddings trained on a large corpus as features of these models. Using bootstrap statistical method, we find that the ensemble of these three models is significantly more efficient in solving this task.

1. Introduction

Language is used by humans not only to specify the emotion we are feeling but also the intensity of the emotion. For example, someone can be slightly angry, extremely sad, moderately happy, etc. The *intensity* of the emotion is therefore a degree or amount of emotion. A system which can recognize the intensity of an emotion can be very useful. For example, it could analyze reviews of a business and know which reviewers were most angrier, the most joyous, and so on.

The importance of this problem was noticed by the SemEval competition creators. The first task of the SemEval 2018 competition is called Affect in Tweets (Mohammad et al., 2018). It has four different subtasks, of which the first subtask is emotion intensity regression. The problem is framed as follows: given a tweet and an emotion E, determine the intensity of E that best represents the mental state of the tweeter - a real-valued score between 0 (least intensive) and 1 (most intensive). The four emotions for this problem are anger, sadness, joy and fear. They were chosen since they are the most basic human emotions.

Our problem is a clear regression problem, since we have to predict the value of intensity, which is a continuous value. We implemented three ML-based models as a solution to this regression problem. The first one is a more traditional Support Vector Regression (SVR) based on RBF kernel. The second one is a deep model based on a Convolutional Neural Network (CNN). The third one is a deep model based on Gated Recurrent Unit (GRU). Features given to these models are lexicon-based features derived from sentiment and emotion lexicons, and word embeddings.

Considering our dataset is relatively small, we use bootstrap re-sampling to produce more examples. These new example sets are used to make statistical tests, and from this tests we can conclude whether the ensemble model is more efficient than the models used for building it.

2. Related Work

The problem of emotion intensity in Tweets gathered attention from NRC Canada researchers. (Mohammad and Bravo-Marquez, 2017a). They created the dataset and experimented with using different features for their model. One of the main features they used are sentiment and emotion lexicons. They used more than 10 lexicons and made a feature vector out of them. Another feature they used is the word embeddings. They were trained from ten million English tweets. They also used other features such as character and word n-grams. The model they used is an L_2-regularized SVR. In their ablation study, they managed to achieve best scores when they used lexicon and word embedding features.

The problem was central part of the WASSA-2017 Shared Task on Emotion Intensity (Mohammad and Bravo-Marquez, 2017b). Twenty-two teams made submissions to this task. The best performing system, *Prayas*, used an ensemble of three different models. The first model is a feed-forward neural network with word embeddings and lexicon features. The word embeddings were trained from 400M Tweets. The second model is the same network, but with weights of the first two network layers being shared by each emotion classifier. The third model is deep learning model formed by LSTM, CNN, max pooling and fully connected layers.

Considering this problem was also present on the SemEval 2018 competition, one of the competitors published their solution to the problem (Jabreel and Moreno, 2018). They used an ensemble of N-Stream CNN and XGboost regression model. For the features, they use emotion lexicons and word embeddings. The word embeddings used are word2vec embeddings trained on 20M tweets, and Glove embeddigs trained on 330M tweets.

3. Dataset

The creation of dataset is explained in the original creator's paper (Mohammad and Bravo-Marquez, 2017a). Considering most of tweets are not associated with the focus emotion, they identified 50 to 100 terms for each emotion and then captured tweets containing that terms. They called these terms query terms.

The dataset for this problem is publicly available at the competition website[1]. The final set of tweets for all for emotions contains 7,097 tweets. This final set is made out of three types of tweets:

[1] https://competitions.codalab.org/competitions/17751

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 55–59, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

1. *Hashtag Query Term Tweets (HQT Tweets)* 1030 tweets with a query term in the form of a hashtag (#[query term]) portion of the tweet;

2. *No Query Term Tweets (NQT Tweets)*: 1030 tweets that are copies of '1', but with the hashtagged query term removed.

3. *Query Term Tweets (QT Tweeets)*: 5037 tweets that include both tweets that contain a query term in the form of a word (no #[query term]) and tweets with a query term in hashtag form followed by at least one non-hashtag word.

Each tweet was annotated by three human annotators. The annotators would get 4 tweets and had to rank them according to the intensity of the focus emotion. Using best-worst scaling, these ranks on thousands of pairs were made into a real number between 0 and 1, which is the intensity of the emotion. The dataset is partitioned into training, development and test sets for machine learning experiments. About 50% of the tweets are in the training set, about 5% in the development set, and about 45% in the test set.

The text of the tweets is not filled just with words but also with emoticons and hashtags. Some studies have shown that emoticons tend to be redundant in terms of sentiment (Go et al., 2009). Similary, hashtag emotion words are also relatively redundant when assessing emotion class. A tweet that conveys anger will still convey anger if we remove the hashtag-words. The dataset creators show that although the emotion is still conveyed if the hashtag is removed, the intensity of the emotion will be lower. Therefore, the hashtag-word emotions are *not* redundant for this problem.

A single entry in the dataset is composed out of four parts: The ID, the text of the tweet, the emotion of the tweet, and a real number between 0 and 1: the intensity of the emotion in the tweet. Several examples of entries are shown in Table 1.

4. Features

There are many different choices when converting raw text into a set of features a machine can process. After a survey of different solutions to this problem, of which the most important ones were given in the last section, we decided to use lexicon features from emotion lexicons and word embeddings.

4.1. Lexicon Features

There are many sentiment and emotion lexicons available to use for tasks involving sentiment and emotion classification and regression. In our systems we use several of these lexicons as features to our first model.

The lexicons consist of a list of common words expressing sentiment or emotion and a score assigned according to sentiment (0 for neutral, 1 for positive and -1 for negative sentiment) or emotion (real number between 0 and 1 assessing the intensity of emotion). Some lexicons have additional features (e.g, variance, different scores for different emotions). We used the sentiment lexicons along with emotion lexicons (which are more suited to our task) since words expressing positive sentiment can be linked to positive emotions (e.g, joy) and vice-versa (e.g, anger, sadness).

An overview of lexicons we use is presented in Table 2. We used a simple procedure to convert a tweet into a lexicon feature vector. For each feature of the lexicon, we go through each word in the tweet and get the lexicon value for it. Then we add all those values and we get a single number for each feature of lexicon. These values from the completed feature vector.

4.2. Word Embeddings

Word embedding is a technique of language modeling and feature learning in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Specifically, it involves mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension.

Word2vec is one of the most popular and efficient model used to produce word embeddings (Goldberg and Levy, 2014). It is based on a neural network architecture and derives embeddings from a large corpus. GloVe is another popular model for producing word embeddings, based on unsupervised learning (Pennington et al., 2014).

We use several pretrained word embeddings. Considering we are working with tweets, we choose the word embeddings with Twitter corpus. The embeddings we used can be found in Table 3. Of these, the most interesting are the Word2vec embeddings trained on 400M Tweets (Godin et al., 2015) and the emoji2vec emeddings trained on 6088 descriptions of 1661 emojis (Eisner et al., 2016).

5. Models

We have tried three models as a solution to this problem. Our first choice is a model based on the Support Vector Regression (SVR) with RBF kernels. The second model is inspired by the deep learning paradigm and is based on a Convolutional Neural Network (CNN). The third model is also inspired by deep learning and is based on Recurrent Neural Network (RNN), specifically its variant Gated Recurrent Unit (GRU). We trained a seperate model for each emotion; for example, for SVR there is a SVR model for anger, SVR model for joy, etc.

5.1. SVR

SVR (Support Vector Regression, also known as SVM-Regression) is a model based on the Support Vector Machine (SVM) applied to the regression problem.

Before feeding the model with data, it needs to be preprocessed. We used TweetTokenizer of the Natural Language Toolkit (NLTK) to tokenize the tweets. We also removed usernames and punctuation. After this, we converted the tokenized text into features: we created a lexical feature vector from lexicons and three word embedding vectors from word embeddings we name WE1, WE2 and WE3, which were explained in the previous section.

These feature vectors and concatenated and fed into *sklearn* implementation of SVR model with RBF kernel (Pedregosa et al., 2011). We use cross validation (CV) on train and development set to find the best hyperparameters

Table 1: Several entries in the dataset for emotion anger

ID	Tweet	Emotion	Score
2017-En-11424	*Circle K is dead to me. I hate your new slushie machine. #livid*	anger	0.750
2017-En-10621	*People often grudge others what they cannot enjoy themselves.*	anger	0.396
2017-En-10112	*I'm just a fuming ball of anger today*	anger	0.708
2017-En-10842	*@savageimiike one of my favorite songs brother, real talk.*	anger	0.157

Table 2: Used sentiment and emotion lexicons

Name	Type	Features
AFINN	Sentiment	1 (positive or negative sentiment with intensity)
Bing-Liu	Sentiment	1 (positive or negative sentiment)
MPQA	Sentiment	1 (positive or negative sentiment)
VADER	Sentiment	2 (positive or negative sentiment with intensity, variance)
NRC-Emotion-Lexicon	Both	10 (8 for emotions and 2 for sentiment)
NRC Hashtag Emotion Lexicon	Emotion	8 (for each emotion)

Table 3: Used word embeddings

Name	Method	Corpus
WE1	Word2vec	400M tweets
WE2	GloVe	2B tweets
WE3	Glove	660K words
WE4	Word2vec	6088 emoji descriptions

C and γ. After training, we give the model the test set and find Pearson's correlation of predicted and true data.

5.2. CNN

Convolutional neural networks (CNNs) can be used in NLP for various tasks. Our second model uses a CNN as a base.

The only preprocessing for this model is the use of Natural Language Toolkit (nltk) TweetTokenizer (Bird et al., 2009). After tokenization, the words are converted into feature vectors by using all of our available word embeddings. For each embedding, we create the CNN part of the network.

The CNN part has three parallel convolution layers, each with 300 filters. The first layer has 1 as the first dimension of its kernel, the second layer has 2, and the third layer has 3. This was done to simulate word 1-, 2- and 3-grams. The second dimension of the kernel is always the length of the embedding vector. After convolution, we use a ReLU function for nonlinearity and then a maximum pooling is applied.

Outputs of these three parallel convolution layers are flattened and concatenated, and then these are concatenated for each embedding. They are then fed into the second part of the network - four fully connected layers using ReLU activation functions, except the last one, which uses sigmoid. The input dimensions of these layers are 2400, 1200, 400, 50. This last layer has one number as output - the predicted emotion intensity.

The model was implemented in pytorch (Paszke et al., 2017) and used RMSprop as optimizer (Tieleman and Hinton, 2012). Early stopping and weight decay was used for regularization.

5.3. GRU

Recurent neural networks (RNNs) are often used in text-related tasks due to their sequential nature. Our model uses a specific variant of RNN called Gated Recurrent Unit (GRU). GRUs are more efficient for most tasks than plain RNNs but are simpler models than Long Short Term Memory networks (LSTMs) (Chung et al., 2014).

As with second model, we use only tokenizing with TweetTokenizer for preprocessing. We then use all word embeddings to produce the embedding vectors. Similar to the second model, we make a GRU part of network for each embedding.

The GRU part of network contains a GRU with hidden size of 512. We feed the embedding vector into this model. We take the last output of the GRU as the output of this part of network.

After this is done for each embedding, we concatenate the outputs and feed it into the second part of the network - one fully connected layer of input size 64 with sigmoid activation. This layer has one output - the predicted emotion intensity.

This model was also implemented in pytorch (Paszke et al., 2017) and used RMSprop as optimizer (Tieleman and Hinton, 2012). Early stopping was used for regularization. Weight decay was not used since the model is very simple so it is not prone to overfitting.

5.4. Fourth model - Ensemble

Our forth model is an ensemble of previous three models. Considering the best teams on WASSA-2017 Shared Task

Table 4: Results for evaluation on the test set

Model	Anger	Sadness	Fear	Joy
SVR	0.7003	0.6764	0.6831	0.6908
CNN	0.7172	0.7148	0.7065	0.7091
GRU	0.6662	0.6760	0.6458	0.7008
Ensemble	**0.7477**	0.7400	0.7311	**0.7392**
Prayas	0.7320	**0.7650**	**0.7620**	0.7320

on Emotion Intensity used ensembles, we made this model.

The ensemble works on the principle of weighted average. Based on each models score (Pearson's correlation value) on the development set, this score is its weight. For example, the model A scores 0.7 on development set, model B scores 0.8 and model C scores 0.6, the prediction for some tweet t will be:

$$ensemble(t) = \frac{0.7 \times A(t) + 0.8 \times B(t) + 0.6 \times C(t)}{0.7 + 0.8 + 0.6}$$

6. Results

6.1. Test set evaluation

Considering the intensity scores for the test set are not available, we used the competition's website[2] to evaluate our models. We managed to achieve third best score in the post-evaluation period.

Our results for each model are presented in Table 4, along with the result of the best team of the WASSA 2017 competition, Prayas (Goel et al., 2017). The scores for each emotion is the Pearson's correlation between predicted and true value. As can be seen, our ensemble model has much better scores than the three models used to build it, and even manages to outperform the winning Prayas system in two emotions: anger and joy.

In introduction we assumed that ensemble model will perform significantly better than SVR, CNN and GRU model alone. As we could not assume normality of the Pearson correlation coefficient, we used bootstrap significance testing (Berg-Kirkpatrick et al., 2012). The ensemble model was shown to be statistically better than the other models.

7. Conclusion

Looking again at WASSA-2017 Shared Task on Emotion Intensity mentioned in the second section, we see that all best teams used ensembles. Knowing that, it comes to no surprise that our ensemble model is statistically proven to be more efficient than the models used for building it.

For future work, we could add more models to the ensemble to increase its efficiency even more. Perhaps after a certain amount of models added it stops being more effective. It's worth noticing that our three models were completely different. It could be that using similar models in the ensemble would not make such an effect on efficiency.

[2] https://competitions.codalab.org/competitions/17751#results

References

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 995–1005, Stroudsburg, PA, USA. Association for Computational Linguistics.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bošnjak, and Sebastian Riedel. 2016. emoji2vec: Learning emoji representations from their description. *arXiv preprint arXiv:1609.08359*.

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12).

Fréderic Godin, Baptist Vandersmissen, Wesley De Neve, and Rik Van de Walle. 2015. Multimedia lab @ acl wnut ner shared task: Named entity recognition for twitter microposts using distributed word representations. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 146–153.

Pranav Goel, Devang Kulshreshtha, Prayas Jain, and Kaushal Kumar Shukla. 2017. Prayas at emoint 2017: An ensemble of deep neural architectures for emotion intensity prediction in tweets. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 58–65.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Mohammed Jabreel and Antonio Moreno. 2018. Eitaka at semeval-2018 task 1: An ensemble of n-channels convnet and xgboost regressors for emotion analysis of tweets. *arXiv preprint arXiv:1802.09233*.

Saif M Mohammad and Felipe Bravo-Marquez. 2017a. Emotion intensities in tweets. *arXiv preprint arXiv:1708.03696*.

Saif M Mohammad and Felipe Bravo-Marquez. 2017b. Wassa-2017 shared task on emotion intensity. *arXiv preprint arXiv:1708.03700*.

Saif M. Mohammad, Felipe Bravo-Marquez, Mohammad Salameh, and Svetlana Kiritchenko. 2018. Semeval-2018 Task 1: Affect in tweets. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, LA, USA.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer,

R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Multi-label Toxic Language Classification of Wikipedia Comments

Lovro Kordiš, Marko Šmitran

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{lovro.kordis, marko.smitran}@fer.hr

Abstract

This paper tackles the problem of classification of toxic text in Wikipedia comments featured in Toxic Comment Classification Challenge. It is a multi-label problem in nature as each comment can fall into multiple categories of toxicity. We approach this task from both classical machine learning and deep-learning angles. In our classical ML approach we used several classifiers, one for binary classification of each category of toxicity. Classifiers were trained on already NLP pre-processed data. We also tried different categories of models – Linear SVM and Logistic Regression. For our deep-learning approach we used models based on LSTMs. Evaluation of models was done on data provided in the challenge. In addition to evaluating models on provided labeled data we further evaluated them with Challenge's official automatic evaluation.

1. Introduction

In the last two decades popularity and usage of the internet exploded. And as the population of users increased, the ways of communication over it also evolved. Today most people in the developed world have access to the internet. In any form of communication between people, situations that might be interpreted as toxic arise. However, perhaps due to the perceived anonymity and physical safety more people engage in toxic behavior online than in real face-to-face communication. A lot of content on the internet is publicly available, and/or stored for a significant amount of time, so any such behavior also has a wider reach. Many places on the internet do not want to condone toxic behavior, so the need to identify such content and perhaps take further action (removal of content, banning the user etc.) is higher then ever. Identification by a human is both inefficient and expensive. So people turned to automatic approach.

In this paper we propose possible solutions to the problem. We focus on identification of toxic content in textual form. Specifically, text in the form of comments from Wikipedia. Although our approaches can be applied to any form of short text. This specific problem was featured in a competition on *Kaggle*[1] ("Toxic Comment Classification Challenge"). Dataset used in this paper is provided by the competition organizers.

Initially we approached the problem from standard machine learning model perspective using SVM and Logistic Regression classifiers for comment classification. Later on we switched to deep-learning model using Bidirectional LSTM in hope of getting better results. Following sections will fully explain all the steps of both approaches. Section 4.1 describes NLP preprocessing steps taken to prepare the data to a satisfactory degree. In Sections 4 and 5 we introduce our models, training and optimization methods used. Section 7 features experimental evaluation of our models and their comparison to both baselines and one another. Finally Section 8 concludes our paper with a summary of our work, insights gathered during our work on the project as well as possible improvements.

[1] https://www.Kaggle.com/c/jigsaw-toxic-comment-classification-challenge

2. Related Work

Problem of identification of toxic texts or similar topics has already been approached from several angles in previous works. Wang and Manning (2012) show that machine learning approach can be effective in NLP problems. Specifically, in (Kennedy et al., 2017) it is shown that ML can be effective in detecting harassment in online communication, topic that is similar to our own.

In the paper with another topic closely related to our own (Sax, 2016), author approaches the problem from a deep-learning standpoint. The author uses several different models (both deep-learning and others) for automatic insult detection. Some of these models use LSTMs and according to their evaluation, these models show great results. In order to tackle the problem with overfitting and to improve results, Lin et al. (2014) propose using Global Average Pooling method, while in (Zhou et al., 2016) we see advantages in combining Convolutional networks with LSTMs.

3. Dataset

We used the training dataset provided by Toxic Comment Classification Challenge on Kaggle.com website. Data consists of Wikipedia comments labeled by human annotators for toxic behavior. Comment consists only of text, no additional information are given (for example prior conversation). Each comment has six distinct labels: toxic, severe toxic, obscene, threat, insult, identity hate. Label value of 1 indicates that the comment belongs to that specific class of toxicity, while 0 indicates that it does not. Each comment can belong to multiple classes. There is a total of 159571 labeled Wikipedia comments in the dataset. Of those, 89.83% are not toxic (they do not belong to any of the above mentioned classes), 9.58% are considered toxic, 0.99% severely toxic, 5.2% obscene, 0.29% are threats, 4.93% insults and 0.88% identity hate. A cursory glance at the data reveals a big imbalance of classes. Imbalance of classes makes training and evaluation of model substantially more difficult.

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 60–63, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Table 1: Label distribution in the dataset

Toxic	Severe toxic	Obscene	Threat	Insult	Identity hate	None
9.58%	0.99%	5.29%	0.29%	4.93%	0.88%	89.83%

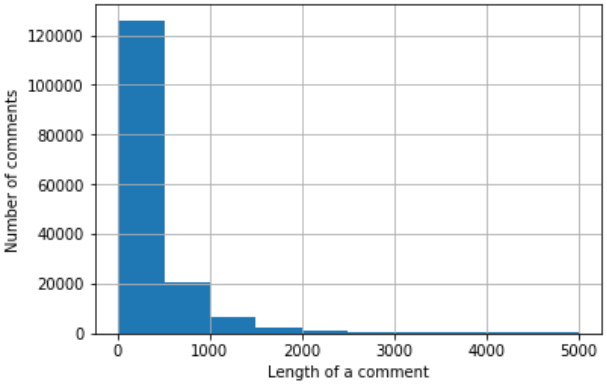


Figure 1: Distribution of comments length

4. Basic Models

The initial task of multi-label classification was made more difficult with the unbalanced dataset (with almost 89 percent of all comments labeled as not toxic). Because of that the baseline was set with over 90 percent accuracy and our main goal was to improve the recall, precision and F1 score metrics.

In this section we describe the process of data preparation and multi-label classification task framing for the SVM and Logistic Regression models.

4.1. Feature Selection

In text pre-processing we only used Snowball[2] stemmer. Lemmatiser was not used as the words in comments can also have lengthened characters or use acronyms and while its often possible to get the original lemma, it comes at the cost of losing information. Features were generated by tokenizing each comment, hashing the resulting n-grams, and computing a TF/IDF value for each token. The resultant feature vectors were used to train a Naive Bayes baseline model, Logistic Regression models and finally SVM model. We used the following features:

- **Unigram and Bigram TF-IDF**: this is a standard feature used in text classification. We used unigrams and bigrams. Bigger ngrams were not used as they provided no more useful information. The size of the dataset meant almost all other ngrams were too rare to have any significance.
- **Named entity recognition**: we used basic named entity recognition with tools provided by NLTK library. We postulated that toxic comments might have some named entities in cases similar to this this example:

  "F##k **Donald Trump** and f##k anyone who still defends him"
- **Language detection**: we used language detection library langdetect. Because the great majority of the comment in the dataset are in English, we supposed foreign language comments might be toxic more often.

To reduce the dimensionality of the feature matrix we also preformed KBest feature selection with χ^2 test on the TF/IDF matrix and selected 500000 best features to use. Named entity recognition (Number of named entities in a comment) and Language detection (A fixed number for every non-english comment) were then added to the reduced matrix as separate columns.

We used NLTK[3] library for all steps except language detection, for which used the langdetect[4] library.

4.2. Multi-label Classification

As previously stated, our task is set as a multilabel classification problem. In machine learning, multi-label classification and the strongly related problem of multi-output classification are variants of the classification problem where multiple labels may be assigned to each instance. Multi-label classification is a generalization of multiclass classification, which is the single-label problem of categorizing instances into precisely one of more than two classes; in the multi-label problem there is no constraint on how many of the classes the instance can be assigned to.

With the absence of useful algorithms that support multi-label classification in the sklearn library, we decided to train 6 separate binary classifiers for each multi-label output. This setup also made the hyperparameter optimization and model evaluation more difficult, as the computing had to be done for every classifier.

5. Deep Learning Models

Multi-label classification is made a lot simpler with deep learning networks, as we do not have to worry about multiple classifiers.

In this subsection we describe the architectures of our several deep learning models that we have created in order to explore the data and to provide context for our results. We also describe the core concepts behind LSTM and Convolutional networks.

5.1. Long Short Term Memory Networks

The main idea of RNN (Recurrent Neural Net), and a big reason of usage in NLP, is in retaining information from

[2] http://snowballstem.org

[3] www.nltk.org

[4] pypi.org/project/langdetect

Table 2: The optimized C parameters for SVM and Logistic Regression models

	Toxic	Severe toxic	Obscene	Threat	Insult	Identity hate
Logistic Regression	128	1	128	64	16	16
SVM	2	0.25	1	1	1	1

Table 3: The optimized parameters of Bi-LSTM and Bi-LSTM + Conv models

	Bi-LSTM	Bi-LSTM+Conv
Dropout	0.15	0.25
LSTM L2 rate	0.001	0.001
BLSTM hidden state	50	40

context of the sentence from the currently processed word to start of the sentence. Despite good results with short sentences, RNN loses its performance with longer text, because of the vanishing gradient problem.

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber (1997). They work tremendously well on a large variety of problems, and are now widely used. Although LSTMs can capture the past context very well, it is desirable to know the future context as well. Bidirectional LSTM solves this by adding another LSTM layer with backwards information flow, which means it can capture the future context.

5.2. Convolutional Networks

CNN networks are famous for their appliance in the Computer Vision domain but have also demonstrated an ability to extract morphological information from word characters, encoding them into neural representations.

5.3. Global Pooling

Dropout and batch normalization can be very effective with feedforward neural networks. However, they can cause problems with RNNs and combined with small dataset can cause overfitting. Thus, we have used pooling methods to stabilise and regularize the training process.

- **Global Average Pooling**: In the last few years, experts (Lin et al., 2014) have turned to global average pooling (GAP) layers to minimize overfitting by reducing the total number of parameters in the model. Similar to max pooling layers, GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor.

- **Global Max Pooling**: Global max pooling is similar to ordinary max pooling layer with the exception of pool size equals to the size of the input.

5.4. Used Models

To test how every architecture handles toxic comment classification we have implemented 2 models based on the following architectures: Bi-LSTM, and a combination of the Bi-LSTM and CNN. We have used pre-trained GloVe word embeddings of vector size 50.

Bidirectional LSTM: The first model is a bidirectional recurrent model with LSTM cells. As input features we use 50 dimensional GloVe (Pennington et al., 2014) vectors trained on the 6B dataset. The Bi-LSTM layer is then preceded by concatenated global max and average pooling values. Fully connected softmax layer is then placed at the networks end.

Bidirectional LSTM with Convolutional layer: A variation of the first model, with Convolutional layer put on top of Bi-LSTM.

6. Model Optimization

We split this section into two parts: basic models and deep learning models parameter optimization.

6.1. Basic Models

As previously stated, because of our task framing, we had to optimize parameters for 6 different binary classifiers. We optimized parameter *C* for **Linear SVC** and **Logistic Regression**. Table 2 shows the optimized parameters for each classifier and model.

6.2. Deep Learning Models

This was a much harder task; because of our limited computing resources and slow training time, we properly optimized Dropout and L2 regularization rates, whereas for hidden layer sizes and batch sizes our method was to test each parameter low and extreme values and then interpolate the best values. Table 3 shows the optimized parameters for each deep learning model.

7. Evaluation and Results

As predicted, all our models have very high accuracy, but the evaluation scores differ on recall, precision and f1 metrics. During the model training and evaluation, we used nested k-fold cross-validation (10 folds) technique with grid search for model parameters. We also compare the results with other solutions at *Kaggle* competition. Our experiments were conducted on an Intel i5 CPU and GTX 1050Ti GPU.

To get an idea of how well our models perform, we used two baseline models. A Naive Bayes classification model and a dummy classifier where we assigned all classes values to 0. The evaluation results shown in Table 4 show we got

Table 4: Evaluation results for our models and baselines

	Accuracy	Precision	Recall	F1
Dummy	0.963268	/	/	/
Naive Bayes	0.971666	0.738822	0.118246	0.188864
LogReg	0.981859	0.655741	0.560120	0.601495
SVM	**0.981954**	0.675585	0.490702	0.552313
Bi-LSTM	0.980480	**0.820342**	0.575812	**0.659872**
Bi-LSTM+Conv	0.980136	0.809786	**0.580549**	0.655871

Table 5: Results of Kaggle competition

Team name	ROC AUC score
Toxic Crusaders	0.9885
Bi-LSTM	0.9804
BI-LSTM+Conv	0.9772
SVM	0.9769
LogReg	0.9767
WeAreRobot	0.9785

the best recall, precision and F1 results with LSTM models, while Logistic Regression and SVM models had higher accuracy. We think the cause of lower accuracy of LSTM models might be due to overfitting on the later epochs of the training process.

Submissions on *Kaggle* competition are evaluated on the mean column-wise ROC AUC. In other words, the score is the average of the individual AUCs of each predicted column. Table 5 shows our models comparison with the winner ("Toxic Crusaders") and an mid-table solution ("WeAreRobot"). Our solution was placed in the upper part of the table

8. Conclusion and Future Work

We proposed 4 different models for solving multi-label toxic comment classification as part of the *Kaggle* competition: Logistic Regression, SVM, Bi-LSTM and Bi-LSTM with Convolutional layer. The provided dataset proved to be somewhat challenging because of the sparsity of toxic comments. We used TF-IDF, Named entity recognition and Language detection features with SVM and Logistic Regression models, and pre-trained GloVe word embeddings for deep learning models. All models had very high accuracy, but deep learning models preformed better in precision, recall and F1 metrics.

For future work, we would experiment with new word embeddings trained only on the toxic comment dataset. We believe it is crucial for word vectors to learn semantic meaning from the domain concrete data because of the toxic comment semantic strucutre. Also, we propose using a different dataset or even creating a new one from Twitter and other sources, which would provide more toxic comment examples.

References

Jennifer Bayzick and Lynne Edwards. 2011. Detecting the presence of cyberbullying using computer software.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. 9:1735–80, 12.

George W. Kennedy, Andrew W. McCollough, Edward Dixon, Alexie Bastidas, John Ryan, Chris Loo, and Saurav Sahay. 2017. Hack harassment: Technology solutions to combat online harassment. *Proceedings of the First Workshop on Abusive Language Online*, pages 73—-77.

Min Lin, Qiang Chen, and Shuicheng Yan. 2014. Network in network.

Andrew McCallum and Kamal Nigam. 2001. A comparison of event models for naive bayes text classification. *Proceedings of the First Workshop on Abusive Language Online*.

Jeffrey Pennington, Richard Socher, and Christoper Manning. 2014. *Glove: Global Vectors for Word Representation*, volume 14. 01.

Sasha Sax. 2016. Flame wars: Automatic insult detection.

Sida Wang and Christopher D. Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, pages 90–94, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. 2015. A c-lstm neural network for text classification. 11.

Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. In *COLING*.

Using NLP Techniques and Model Selection to Improve Performance of SMS Spam Classification

Marin Krešo, Matteo Miloš, Josip Renić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{marin.kreso,matteo.milos,josip.renic}@fer.hr

Abstract

Recent reports clearly indicate dramatic growth in volume of SMS spam messages. SMS spam classification is a challenging problem, as this kind of messages are rife with idioms and abbreviations. Most common and baseline solution for this is using Multinomial Naive Bayes algorithm with Bag-of-words term frequencies as features. As an alternative, we propose pipeline approach that uses NLP (Natural Language Processing) techniques, extracts new features and does hyperparameter optimization for most popular machine learning classification algorithms. Our results on the SMS Spam Collection dataset show that by incorporating our proposed pipeline approach, SMS spam classification system can yield statistically significant performance gain as compared to the baseline.

1. Introduction

SMS, which stands for short message service, is one of the simplest and most popular ways of communication between people. It is a communication service component of phone, web or mobile communication systems and it uses standardized communication protocols that enable mobile devices to exchange short text messages. Although SMS messages were initially designed only to send textual content, today we can send image, video and audio content as well. While SMS already is a huge market, it is still growing, although there is numerous competition that use IP-based messaging service, such as WhatsApp, Viber, iMessage etc. One of the reasons SMS messaging is still so popular is because of its cheap price. For example, an actual cost of sending and SMS in Australia was approximately $0.00016 in 2015, and today it is even cheaper. However, industry earning does not suffer despite cheap prices, because there are, according to Portio Research, approximately 16 million SMS sent per minute. In addition to regular usage, for day-to-day informal communication between people, SMS has also become a massive commercial industry. At the end of 2016, there were about 37 million consumers that opted-in to receive business SMS, and it is predicted that by 2020 that number will reach 49 million. Despite all the benefits SMS communication offers, with years and price of messaging going down, some downsides also came up. The main downside we can highlight is emerging phenomenon of SMS spam. As SMS spam we can classify any message that wasn't wanted and solicited from the user. One of the things that contributes to growth of SMS spam is already mentioned cheap price of SMS, so it is almost costless for spammers to send malicious messages. Except the main reason, which are annoying and potentially dangerous messages, one of the problems is that some carriers still require from their users to pay to receive text messages. Also, there are numerous possibilities for financial frauds, and reportedly, in 2016 Canadians more than $600000, which is about 5 times more than 2 years earlier. All mentioned problems with today's SMS communication leaded us to designing classification algorithm, that can be used in classifying the messages either to ham or spam messages. While we already briefly explained what spam messages are, ham messages we can describe as the exact opposite, because they are regular messages, either formal or informal, that user is happy to receive. Our main goal was to improve results achieved by using Multinomial Naive Bayes, which we used as a baseline model for our approach. While many previous studies oriented purely on machine learning algorithms, our approach included using the NLP techniques. To produce statistically better results with our model, we incorporated NLP techniques such as stemming, stop words removal, feature extraction and tokenization This paper will be organized in following sections. In Section 2 we will discuss previous works on this matter, Section 3 offers us overview on data we are going to use. In Section 4 we will present our methodology that will be used for classification and in Section 5 we are going to conduct actual experiment. In the last section, finally we are going to discuss conclusion and potential for future work.

2. Related Work

Through the recent years, there were multiple attempts that tried to develop various techniques for successful classification of SMS spam and ham messages. Everything started in 2004, when Xiang and Ali (2004), decided to used Support Vector Machines for classification. Same method was used by Gomez Hidalgo et al. (2006), with addition of selecting tokens using Information Gain. Next step forward was in 2009, when Duan et al. (2009) used k-NN (k-nearest neighbor) algorithm along with other spam detection methods. Almeida et al. (2011) used 13 different classification algorithms on a dataset that contained 5500 SMS messages aggregated from multiple sources. They concluded that the best performing algorithm was an SVM combined with alphanumeric tokenization. Throughout the years, although the results were good, few limitations were recognized. Despite of high accuracy, there was noted also a high false-positive rate and a low efficiency, which is caused by using large number of input features. Our goal was to address these limitations and try to improve the results, either by

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 64–68, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

optimizing algorithm such as SVM, or using some completely different classification algorithm.

3. Data

As in any scientific research, we had to make sure we have reliable and representative data. Because there is lack of quality SMS spam datasets, we decided to use one which was created for previous scientific work (Almeida et al., 2011). That dataset, which is publicly accessible at KAGGLE[1], is derived from multiple trustworthy sources.

Finding trustworthy SMS spam messages is a lot harder job than finding the ham ones. Therefore, as a first source, they used The Grumbletext Web site[2], which contained 425 manually classified spam messages. Grumbletext is a UK forum in which cell phone users make public claims about SMS spam messages. However, most of them do that without reporting the very spam message received, so identification of spam messages was very hard and time-consuming task. Their next source was, NUS SMS Corpus[3], dataset od 10000 ham messages collected for research at the Department of Computer Science at the National University of Singapore. A subset of 3375 SMS ham messages was randomly chosen, and all the contributors to the dataset were made aware that their contributions were going to be made publicly available. Also, a list of 450 SMS ham messages was collected from Caroline Tag's PhD Thesis[4]. As as a final source, authors incorporated the SMS Spam Corpus v.0.1 Big[5]. That is a dataset that contains 1002 ham and 322 spam messages.

Using all mentioned sources, we can conclude that our dataset contains a total of 5574 messages, of which 4827 are ham, and the rest, 747, are spam messages. As far as our knowledge goes, that dataset is the largest SMS corpus available, that contains both spam and ham messages, and therefore we determined that it is representative example for our research.

4. Proposed Methodology

Standard approach is to use Multinomial Naive Bayes with Bag-of-Words model for spam/ham detection. This model is expected to produce good results without any NLP and hyperparameter optimization. Goal of our system is to produce statistically significant better results with two different tokenization approaches. Figure 1 shows steps involved in each system.

4.1. Tokenization

Due to limited SMS length people compress their messages. SMS is filled with colloquial and slang speech e.g., "Nah", "k", "lol", etc. From 13654 unique tokens observed over whole dataset only 24.06% tokens appears in set of English words from NLTK[6] Wordlist Corpora.

[1] http://www.kaggle.com
[2] http://www.grumbletext.co.uk/
[3] http://www.comp.nus.edu.sg/~rpnlpir/downloads/corpora/smsCorpus/
[4] http://etheses.bham.ac.uk/253/1/Tagg09P hD.pdf/
[5] http://www.esp.uem.es/jmgomez/smsspamcorpus/
[6] http://www.nltk.org

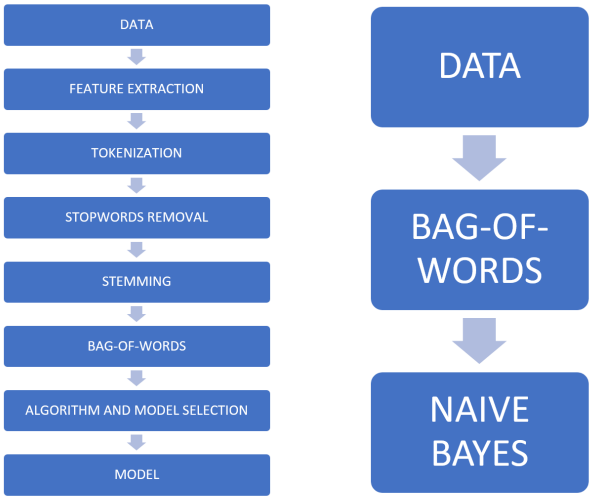


Figure 1: Proposed system pipline variations.

People tend to lose formal grammatical structure and misspell words when writing SMS. This finding conveys us to peruse two different tokenization approaches, first standard tokenizer and second one to tackle misspelled words:

- tok1 - tokens created by white space elimination, for sentence "Lol your" we would get tokens "Lol" and "your".
- tok2 - tokens created from characters from lowercased sentence with n-gram range of (2,5), for sentence "Lol your" we would get tokens 22 tokens, e.g. "lo", "lol", "lol ", "lol y", etc.

For tok2 n-gram range (2,5) was chosen as best performing n-gram range after extensive search.

4.2. Stop Words

After tokenization with *tok1* tokenizer, we analyzed the most common tokens which turned out to be stop words. As well, for spam and ham messages the most common tokens are stop words. Due to nature of spam massages we would expect the most common tokens to be "call", "free" and "win". Next step in our system pipeline is stop words removal, which can only be performed with *tok1* tokenizer. We used standards NLTK corpus stop words. We can see the most common tokens after stop word removal in Table 1.

4.3. Stemming

Words appear in natural language in many forms:

- Inflection - adding a suffix to a word, that does not change its grammatical category, such as tenses in verbs (-ing, -ed, -s), plural in nouns (s).
- Derivation - adding a suffix to a word, that changes its grammatical category, such as nation (noun) = national (adjective) = nationalize (verb).

Table 1: 10 most common tokens

all data	spam pruned	ham pruned
.	call	im
to	free	im
I	2	2
you	txt	get
	u	ltgt
?	ur	ok
!	mobile	go
a	text	ur
...	4	got
the	stop	ill

The goal of stemming is to reduce such forms of words to their common base form, which could be useful for our task of classification of spam in SMS messages. Most of the current spam filters use keywords to detect spams. These keywords can be misspelled, even on purpose to avoid detection. It is impossible to predict all possible misspellings for some keywords, so that is where our stemmer comes in handy. It is important to notice that stem of some word does not need to be identical to the morphological root of the word. It is usually sufficient that related words map to the same stem, even if the stem itself is not a morphologically valid root. Stemming is typically done by suffix stripping plus some extra steps and checks.
Algorithms for stemming have been studied in computer science since the 1960s. For our paper, we are going to use on the most popular stemming algorithms - Porter Stemmer. Porter's algorithm consists of 5 phases of word reductions :

- Phase 1 deals with plurals and past participles.
- Phase 2-4 deal with derivation of words.
- Final phase for tidying up.

Another approach to solve the issue with many forms of words is lemmatization, but we are not considering this in our paper.

4.4. Feature Extraction

Due to SMS limited size and average length being 80 characters, given that largest observed SMS has 910 characters, we could benefit from additional information in form of hand-crafted features. In next sections we will explain the proposed features. Besides proposed features we explored additional ideas for features as: number of punctuations, number of capital words, number of lowercase words and ratio of correct English words. These features did not improve performance and their presence in ham and spam was equal.

4.4.1. Length

Spammers usually use longer messages: average spam length is 139 while average ham 72 characters. Our explanation is that spam messages are context free, there is no previous conversation, so they need more words to convey their message, while a ham message does not suffer from lack of context. Figure 2 shows us histogram of spam and ham length histogram. We can see a clear separation between ham and spam in length, which proves that length would be good feature candidate.

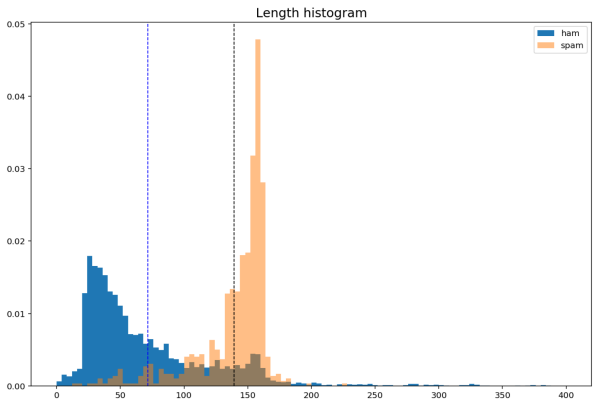


Figure 2: Spam and ham character length histogram with corresponding averages.

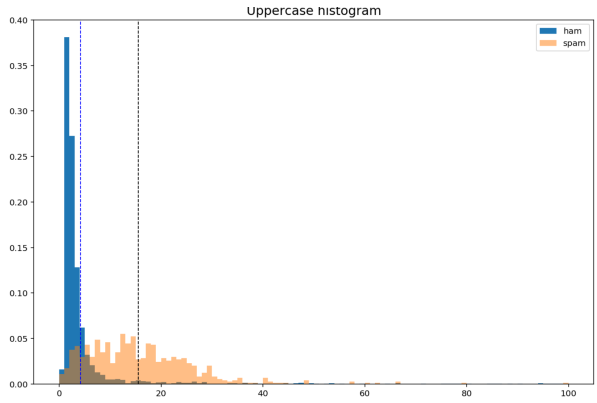


Figure 3: Spam and ham uppercase character histogram with corresponding averages.

4.4.2. Number of Upper Characters

After observing spam messages, we found one typical spam characteristics: usage of large amounts or uppercase characters. Spam messages tend to include all capital words like: "WINNER", "WIN", "FREE","CALL" and "URGENT". There are exceptions in ham messages that contain all capital words which tends to be common in SMS when people express urgency or excitement. Average number of upper characters in ham is 4.17, and in spam is 15.48. Figure 3 shows us histogram of spam and ham upper characters histogram. We can see a distinction for large amounts of spam messages while ham messages tend to contain very small to none number of upper characters.

4.4.3. Number of Numeric Characters

When we think of SMS spam we think of typical prize money award message that wants us to call some number. To cover those messages, we introduce number of numeric characters feature. Average numeric characters in spam

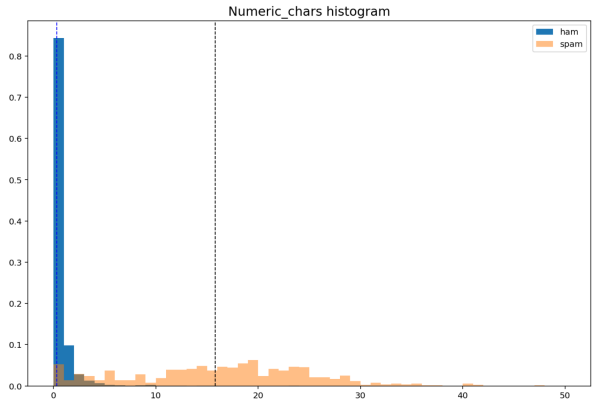


Figure 4: Spam and ham numeric character histogram with corresponding averages.

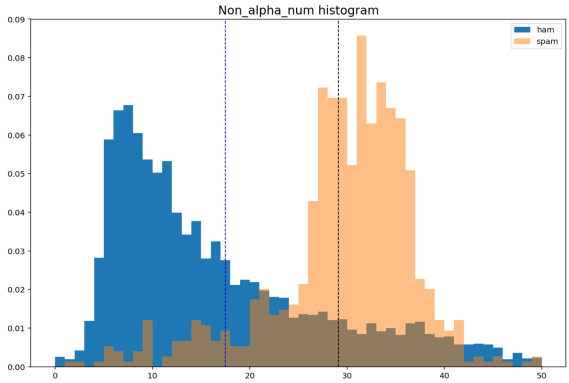


Figure 5: Spam and ham non-alphanumeric character ratio histogram with corresponding averages.

messages is 15.81 and for ham messages is 0.3. We can see clear distinction between ham and spam messages. Figure 4 shows us that almost all ham messages contain some to none numeric characters while most of spam messages contain a lot more.

4.4.4. Number of Non-Alphanumeric Characters

Spam messages tend to use more special characters. One explanation is that they need to be more interesting and flashy. On the other hand, people sending ham messages need to convey their message in limited amount of space and do not use non-alphanumeric characters a lot. As well some mobile phones have limited support for special characters. Ham average number of non-alphanumeric characters is 17.5 and for spam is 23.1. Figure 5 shows clear bimodality in non-alphanumeric characters distribution which supports our claim as a good feature.

4.5. Bag-of-Words

Bag-of-words is simplifying unstructured representation of text as an unordered set of terms (text is considered as a 'bag' containing words). While disregarding grammar and word order, this model considers the words and their frequency of occurrence in text.

In practice, the Bag-of-words model is mainly used as a tool of feature generation. Machine learning algorithms cannot work with text directly, and this model converts text into vectors of numbers representation which is suitable for algorithms we are going to use. These vectors are sparse (most elements are zero), especially for our problem that has huge vocabulary across messages in collection. It is beneficial and often necessary to use specialized algorithms and data structures that take advantage of the sparse structure of such vectors. We are going to use such vectors as features in our models.

4.6. Machine Learning Algorithms

4.6.1. Baseline

We are going to create simple but effective Naive Bayes baseline model that will be used for comparing with other, more advanced algorithms that use NLP techniques and are optimized by using grid search and cross-validation (model selection). For proof that some other classifier is better than baseline we will perform statistical significance test.

Naive Bayes is a classification algorithm which uses data about prior events to estimate the probability of future events based on applying Bayes' theorem. It has a strong (naive) assumption that every feature is independent of others given the class label. Probabilistic classifiers like this are one of the oldest solutions for spam filtering. They are typically used with bag-of-words model features to identify spam.

It is often used as baseline model due to its simplicity and often good performance. It is simple and fast to train, it has no hyperparameters that need to be tuned, it has a probabilistic output, they have been demonstrated to be reliable and accurate in number of applications of NLP, so there is no confusion about popularity of this algorithm.

Naive Bayes model assumes that each of the features it uses are conditionally independent of one another given some class. More formally, if we want to calculate the probability of observing features *f1* through *fn*, given some class *c*, under the Naive Bayes assumption the following holds:

$$p(f1, ..., fn|c) \propto p(c)p(f1|c)...p(fn|c)$$

We have said nothing about the distribution of each feature. In other words, we have left $p(fi|c)$ undefined. Multinomial Naive Bayes is a specialized version of Naive Bayes that is designed more for text documents. It simply lets us know that each $p(fi|c)$ is a multinomial distribution, rather than some other distribution. This distribution works well for data which can easily be turned into counts, such as word counts in text, so we can apply it in our problem.

4.6.2. Candidates

The "No Free Lunch" theorem states that there is no one model that works best for every problem. Maybe there are better solutions than our baseline model which sometimes isn't powerful enough due to its independence assumption, and it has problems with high dimensionality of the feature

	bow	tok1	tok2
MultinomialNB	0.9714*	0.8945	0.9733
LogisticRegression	**0.9733**	0.9733	0.9820
SVC	0.9586	0.9664	0.9702
KNeighborsClassifier	0.8092	0.9681	0.9799
RandomForestClassifier	0.9608	**0.9873**	**0.9849**
XGBoost	0.9443	0.9864	0.9840

Table 2: Experiment results with $F_{0.5}$ scores

space. To test if there is some better machine learning algorithm than baseline, we are going to use some of the most common machine learning algorithms for classification:

- Logistic Regression
- Support Vector Machine
- k-NN
- Random Forest
- XGBoost

5. Experiment

For our experiment we evaluated three different approaches. One originally proposed system with Bag-of-Word only and two our systems with *tok1* and *tok2* tokenizers. When using *tok2* tokenizer stop words removal and stemming steps were omitted. We have done 75/25 stratified train/test split of our dataset. For each model we tuned their hyperparameters with 10-fold cross validation method on train set. All models are evaluated on test set.

5.1. Evaluation

Due to specific nature of spam and ham messages, emphasis was put on classifying ham messages correctly. We concluded that it would be fine if we had some spam messages misclassified but much worse if ham message was labeled as spam. Spam messages present positive examples and ham messages are negative examples. Due to that fact we want to prefer models that have less false positives. As a result, we choose $F_{0.5}$ score as a performance measure to put emphasis on precision measure, which will give us what we want.

5.2. Results

First column in Table 2 represents results with Bag-of-Words approach which was implemented as CountVectorizer. We can see that tuned Logistic Regression produces slightly better performance than our baseline model.

Second column represents results of our proposed system with *tok1* tokenizer which includes all extracted features and NLP preprocessing techniques. We can see increase in performance in most of our models. Worth mentioning is sharp decline in baseline performance which shows us that this model does not benefit from this features for this task.

Third column represents results of our proposed system with *tok2* tokenizer which also includes extracted features. In this approach we gain 20 times more features from text. Most models had increase in performance slightly, but our best performing models Random Forest Classifier had slight decrease. We concluded that tradeoff between performance increase and explosion in feature dimensionality goes in favor of *tok1* tokenizer.

5.3. Significance Testing

Lastly, we tested our baseline model against best performing model within our system which is Random Forest classifier. Significance testing was done with one sided paired t-test on 99% confidence level, and alternative hypothesis that our approach with Random Forest classifier performs better then baseline model. Performance was measured on whole dataset with nested cross validation with 30-fold both on inner and outer loop. Observed p-value was 0.0025 or 0.25%, which indeed proves that our system has statistically significant increase in performance over baseline model.

6. Conclusion and Future Work

In this paper, we propose a pipeline approach that combines NLP techniques such as stemming and stop words removal, with model selection of most popular machine learning algorithms for classification. We used SMS Spam Collection dataset which is the largest SMS corpus as we know. We conduct experiments on this dataset of comparing baseline model (Multinomial Naive Bayes) with best solution from our pipeline approach. We showed that our pipeline approach can yield statistically significant performance gain as compared to the baseline. In future, we would like to explore the deep learning models, especially LSTM.

References

T.A. Almeida, J.M.G. Hidalgo, and A. Yamakami. 2011. Contributions to the study of sms spam filtering: New collection and results.

H. W. Brown, A. S. Forbes, and S. D. Smith. 1900. Title title title title title title title title title. *Journal journal journal*.

L. Duan, N. Li, and L. Huang. 2009. A new spam short message classification. 2:168–171, March.

Jose Gomez Hidalgo, Guillermo Cajigas Bringas, Enrique Sanz, and Francisco García. 2006. Content based sms spam filtering. 2006:107–114, 01.

Chowdhury Morshed Xiang, Yang and Shawkat Ali. 2004. Filtering mobile spam by support vector machine. *CSITeA'04: Third International Conference on Computer Sciences, Software Engineering, Information Technology, E-Business and Applications, International Society for Computers and Their Applications (ISCA)*, pages 1–4.

Personalized Medicine: Redefining Cancer Treatment

Marin Kukovačec, Toni Kukurin, Marin Vernier

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{name.surname}@fer.hr

Abstract

With the recent advances in DNA sequencing, genetic testing is becoming an ever-more important aspect of personalized medicine. However, progress in this area has been slow due to significant amount of manual work still required to understand genomics. This paper approaches a Kaggle competition task of automatically classifying genetic mutations that contribute to cancer tumor growth (*drivers*) in the presence of neutral mutations that do not affect the tumors (*passengers*). Based on compiled abstracts of medical articles we are asked to accurately distinguish between 9 classes of mutation effects of the discussed genes. We show that this task can be approached without introducing significant domain knowledge to the classifiers, and that simple NLP pipelines can work relatively well in the proposed setup.

1. Introduction

Cancer develops as a consequence of a change in the DNA sequence of a genome. The genetic mutations that infer the cancer growth are called the *drivers*, while the other, neutral, mutations are called the *passengers*.

An emerging approach in medicine named *precision medicine*, takes into account data and habits of each patient to determine the disease treatment (De and Ganesan, 2017). This approach is possible even in cancer treatment, due to massive advances in genome sequencing. Improved genome sequencing has enabled categorizing the mutations of all the major tumor types as *drivers* and *passengers*. This categorization is usually done by experts and clinical pathologists who annotate reported mutations based on the evidence from text-based clinical literature. It is a tedious and time consuming process. In principle, machine learning algorithms could be used to make the task easier and more efficient by automatically classifying those mutations.

For this purpose, Memorial Sloan Kettering Cancer Center (MSKCC) launched a competition on Kaggle, providing a knowledge base with thousands of mutations, annotated by world-class researchers and oncologists. The overall goal of this competition is to *take Personalized Medicine to its full potential*.

2. Related work

Despite the fact that clinical free text (primary data about patients, fragmentary notes by doctors, etc.) could provide a lot of practical data to use in the stated problem of personalized medicine, these texts create significant technical problems for natural language processing models (Pestian et al., 2007). The reason behind this is that such texts are often truncated and unstructured. On the other hand, texts provided in our dataset (clinical literature) are complete sentences and have defined structure. This makes it easier for the machine learning models to perform well. A good insight to the precision medicine approach can be found in the work of Lee et al. (2018) in their paper about the treatment of acute myeloid leukemia, but their work is not focused on machine learning. However, they do utilize a probabilistic graphical model to classify the mutations and make their assignment easier.

An extensive piece of work that focuses on machine learning can be found in (Munoz, 2017). It provides a comparison of a number of classical machine learning and deep learning models. It shows that the models that use TF-IDF features achieve the best results. This paper was the starting point in our work tackling the problem of personalized medicine.

Finally, Avdeeva (2017) tried to incorporate genes and gene variations as features in her work but ultimately concluded that they do not improve performance. Her analysis finds their combinations are unique and therefore fail to provide significant information gain by themselves.

3. Data

The data we used was provided by the Memorial Sloan Kettering Cancer Center (MSKCC). They published it as part of a competition they organized for the NIPS 2017 Competition Track. The competition was held on Kaggle.[1]

The provided dataset is a knowledge base manually labeled by world-class researchers and oncologists. It consists of 3321 training samples and 986 test samples multilabelled with 9 different classes. The classes represent types of genetic mutations. The experts used abstracts of existing clinical texts to infer these mutations. As shown in Figure 1, the classes are extremely unevenly distributed.

We also note that the 986 competition test data samples actually consist of two separate (*public* and *private*) datasets. The public dataset consists of 749 samples (76%), while the private one consists of 237 samples (24%). The reason for this split is that the competition organizers wanted to prevent competitors from manually labeling test instances, so the *private* dataset is actually comprised of papers published during the two months that this competition was ran.

Another thing to mention is that only part of the 986 test samples consist of *real* data, the rest being machine-generated. Only *real* data inputs are scored during evalua-

[1] https://nips.cc/Conferences/2017/CompetitionTrack

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 69–71, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

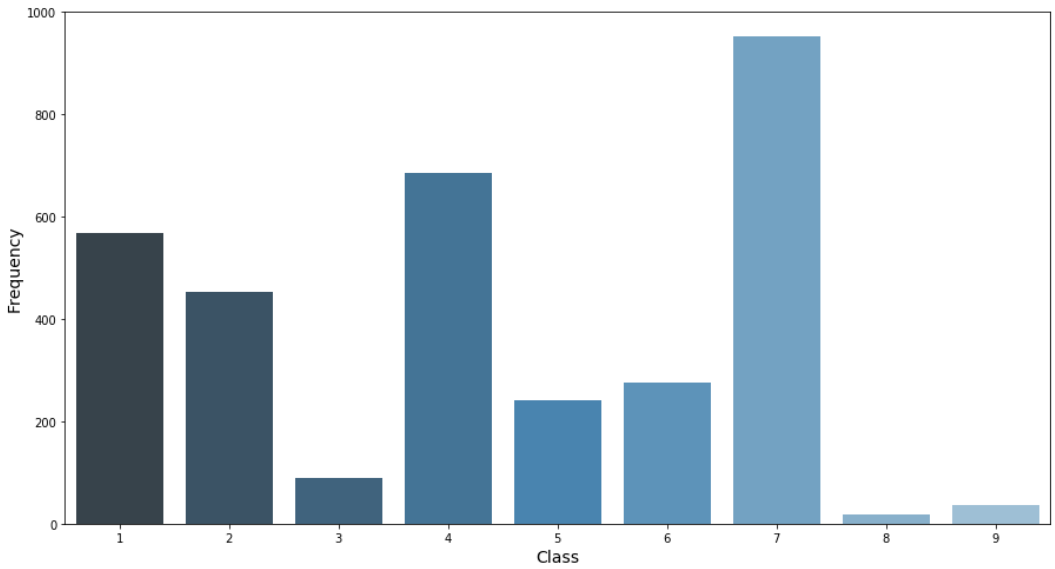


Figure 1: Distribution of genetic mutation classes in train data

tion by the Kaggle platform, and the organizers do not disclose any further information. Based on comments made by other competitors, we speculate there to be between 100 and 200 *real* datapoints, most of them in the *private* dataset.

Besides natural language, additional two inputs that were provided by the authors contain meta-information about the texts (*genes* and *variations* discussed in all of the merged clinical texts).

4. Model

As our evaluation baseline, we decided to setup a simple bag-of-words model. The feature set consists of binary values of word appearances, and we train it using a multinomial Naive Bayes. Even though Naive Bayes does not provide probabilistic outputs, we find that – since most of the training data is classified into a few classes at most – it nevertheless appears sensible to use it as a lower-bound.

For our first model we tried improving upon the baseline by incorporating TF-IDF values. We also did standard text preprocessing by way of converting it into lowercase and removing stopwords. We then fed this to a probabilistic SVM classifier. As we expected from (Munoz, 2017), this drastically improves the results compared to the naive word-count approach.

For our second model, we derive features using our own custom resources along with gene and variation data from the dataset:

Uni-gram – counts how many times in text a unigram from each class occurred,

Bi-gram – counts how many times in text a pair of words occurred,

Gene – tries to assign genes to the correct class,

Variation – tries to assign variations to the correct class

The last two mentioned features performed poorly and due to our lack of familiarity with the medical domain we unfortunately were not able to reason about their relative importance; in the end, we did not manage to include them in a way which would improve model performance. That said, it is curious to note that using a custom count-based bag-of-words approach actually manages to outperform all others by a fairly large margin. We go into more detail about this in the next section.

5. Evaluation

This is a multi-labeling task, and the official evaluation metric proposed by the competition organizers is log-loss which we also use to score our results.

Despite our training dataset being fairly small, we have done 5-fold cross-validation and then evaluated our models on the official submission dataset. Results of our experiments are shown in Table 1. Private score of log-loss is always higher than public one despite the fact that it is evaluated on approximately 24% of all of the test data, while the public score is evaluated on the remaining 76%. As mentioned earlier, we believe that most real data (which is used to score the results) is located in the private dataset and that this explains the higher log-loss on private dataset.

Both of our models outperformed the baseline by a significant amount. SVM model derived from TF-IDF vectorizer values scored a private dataset result of 4.961 and public of 1.934. We then performed experiments on the model derived from our own features. We first tested a model derived from gene and variation features. It performed worse than the TF-IDF model, acquiring a log-loss of 9.455 on the private and 6.446 on the public dataset. Because of that poor performance, we decided not to use them in further experiments. Other two features were unigram and bigram features. Since there were only two of them, we were able to isolate each and perform an ablation study. SVM model with custom unigram feature performed the best if you consider private dataset. It has log-loss 2.823 in private data and 1.951 on public data. On the other hand, bigram feature had a better public score with log-loss of 1.811, but much

	Cross-validation	Private score	Public score
NB baseline	22.599	23.000	13.092
TF-IDF	4.562	4.961	1.934
Gene, variation	8.823	9.455	6.446
Unigram	**1.878**	**2.823**	1.951
Bigram	3.983	4.068	1.811
Unigram, Bigram	3.786	3.802	**1.809**

Table 1. Overview of log-losses

worse private score of 4.068. The combination of these two features gave us an improvement on public score with log-loss of 1.809, but poor private score with log-loss 3.802. So as shown in table 1, our best score on private board was scored by model with only unigram feature, while the best public score was achieved with model derived from both unigram and bigram features. Overall, it seems that the custom unigram model performs best across all datasets.

6. Conclusion

With the ever-increasing advances in NLP, personalized medicine is becoming an interesting avenue for helping doctors perform their day-to-day work. While automated information extraction for a high-risk domain such as medicine is still too underdeveloped to be used in practice, we have demonstrated that even fairly basic NLP approaches achieve positive results. It is not infeasible that personalized medicine could provide some assistance with patient diagnostics in the future.

Since our approach mostly relies on extracting natural language features without guidance, we believe an interesting avenue for future work might be trying to deduce some more informative data from texts with the help of clinical pathologists. We believe the features that did not work well for us (i.e., gene and variation information) could prove to be usable if their implementation in an NLP pipeline was guided by a domain expert.

References

Ekaterina Avdeeva. 2017. Cancer. all 'gene + variation' values are unique. https://www.kaggle.com/eavdeeva/cancer-all-gene-variation-values-are-unique/notebook, jul.

S. De and S. Ganesan. 2017. Looking beyond drivers and passengers in cancer genome sequencing data. *Annals of Oncology*, 28(5):938–945.

Su-In Lee, Safiye Celik, Benjamin A. Logsdon, Scott M. Lundberg, Timothy J. Martins, Vivian G. Oehler, Elihu H. Estey, Chris P. Miller, Sylvia Chien, Jin Dai, Akanksha Saxena, C. Anthony Blau, and Pamela S. Becker. 2018. A machine learning approach to integrate big data for precision medicine in acute myeloid leukemia. *Nature Communications*, 9(1):42.

Jorge Munoz. 2017. Personalized medicine: Redefining cancer treatment with deep learning, Nov.

John P Pestian, Christopher Brew, Paweł Matykiewicz, Dj J Hovermale, Neil Johnson, K Bretonnel Cohen, and Włodzisław Duch. 2007. A shared task involving multi-label classification of clinical free text. In *Proceedings of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing*, pages 97–104. Association for Computational Linguistics.

Littlest Teeniest Tiniest Deep Architecture for Detecting Emotion Intensity

David Lozić, Luka Markušić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{david.lozic,luka.markusic}@fer.hr

Abstract

This paper describes the system we used to find a lightweight deep learning model which offers acceptable results with fast performance. To do this we constructed a basic grid search framework which was trained on the English version of the 2018 SemEval Task 1 Emotion Intensity Regression dataset. This procedure involves both architecture and hyperparameter search of convolutional models. The goal of this work was to find the simplest model with the least number of parameters that provides satisfactory performance. We successfully managed to realize this goal in form of a small model with a single convolutional layer with 10 filters and kernel size 3.

1. Introduction

Sentiment analysis might be more commonly known by its flashier and broader title, *opinion mining*. Generally speaking, the task of sentiment analysis is to identify and categorize the writer's feelings and/or opinions into several categories. It has become critical for companies to see how customer's opinions change with the passage of time and trends, and in the academia, the very popular SemEval series hosts many sentiment analysis competitions annually, with many different subtasks.

In recent years, the trend seems to follow deep learning models, as they often occupy the top positions on the leaderboards. At the very top of these competitions many elaborate models and complex ensembles can be found which often provide a minor boost in performance. Given that these models provide satisfactory performance, we are interested in finding the minimal working model architecture which still offers acceptable results. The motivation behind this is to find the optimal industry oriented model which would work well in a production environment.

This is made possible due to the rather small size of the dataset, which provides us with the unique opportunity to do a detailed architecture and hyperparameter analysis. In essence, we perform an architecture and hyperparameter grid search in pursuit of a lightweight model.

We used the 2018 SemEval Task 1 Emotion Intensity in our research. The task was comprised of several different problems:

EI-reg: given a tweet and an emotion, determine the intensity of that emotion as a score between 0 and 1.

EI-oc: given a tweet and an emotion, classify the tweet into one of four ordinal classes of intensity.

V-reg: given a tweet, determine the valence of a tweet as a score between 0 and 1.

V-oc: given a tweet, classify it into one of seven ordinal classes.

Our main focus point was the EI-reg task.

2. Related work

A substantial number of approaches rely greatly on an underlying sentiment lexicon (Kolchyna et al., 2015) which is one of the most robust methods, but it does not yield significant improvements unless multiple lexicons are used in tandem. In such a scenario with multiple lexicons, the problem of word matching arises due to the fact that the twitter corpora is intertwined with syntactic mistakes which the lexicons are not equipped to deal with. Our main source of inspiration was EiTAKA's system (Jabreel and Moreno, 2018) for SemEval-2018. Their implementation uses multiple word embeddings, multiple semantic lexicons, a large convolutional model with multiple channels combined with an XGBoost regressor into an ensemble. Multiple semantic lexicons were used to deal with the previously mentioned lexicon problem.

Their work was focused towards reaching the best results, as for our proposed model builds upon their work, trying to strip down all the seemingly unnecessary material while preserving performance.

3. Resources

The dataset for all of the tasks was provided by SemEval itself, both in English, Spanish and Arabic, however, our language of choice was English. Additional resources in form of word embeddings and sentiment lexicons were provided by Stanford (Pennington et al., 2014) and National Research Council in Canada (Jabreel, 2016), respectively.

In order to stay in the mindset of trying to obtain a lightweight model, we have limited the word embeddings to 200-dimensional vectors, and only one sentiment lexicon will be used.

4. System Description

In this section, we present how our system performs the grid search. The system has been implemented using the Keras deep learning library (Chollet and others, 2015).

4.1. Preprocessing

For text preprocessing, we use the tokenizer tool provided in the Keras library. The tweets are lowercased and the special characters and punctuations are removed. These characters include: *!"$%()*+,-./:;¡=¿?@[\]^_‘{—}* .

After this normalization step, the tweets are tokenized using the Keras tokenizer.

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 72–74, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

4.2. Input

The words from the preprocessing step are mapped to word embeddings and affect feature vectors. For word embeddings, we use the GloVe word vectors (Pennington et al., 2014) which embed words into 200-dimensional vectors trained on the Twitter corpora, on 2B tweets in total. The affect feature vectors map words to the affect values which are extracted from the NRC affect lexicon (Jabreel and Moreno, 2018) which were also trained on the Twitter corpora. In case of absent words, the corresponding feature vector equates to the average affect intensity across the entire lexicon. The lexicon consists of 6000 words in total, and each word may represent multiple emotions. These features produce 4-dimensional word vectors where every dimension represents an affect intensity value for the following emotions: *anger, fear, joy, sadness*. Finally, the embeddings and the feature vectors are concatenated into a single 204 dimension embedding. These embeddings represent the model input.

4.3. Architecture grid

The system performs hyperparameter search for all architectures given by the user. An architecture is defined by a list of layers from which the model is built using the *Sequential* Keras API. An example architecture can be seen in Figure 1. The system fills every layer of the architecture with arguments generated from the hyperparameter grid. The sequential list structure of architectures does not allow for more complex models with multiple inputs that can be compiled using the *Functional* Keras API. But this was not our goal since we are searching for minimal models, although the system could be easily modified to satisfy such a need.

4.4. Hyperparameter grid

To perform hyperparameter search, the user needs to define a list of possible values for chosen parameters of every type of layer which is present in the architecture. The system creates a Cartesian product of hyperparameters which are used to fill the layers of the specified architecture.

4.5. Training

The model has been trained using the Adam optimizer with the following values: `Adam(lr=0.001, beta1=0.9, beta2=0.999, epsilon=None, decay=0.0, amsgrad=False)` Since the dataset is small, the optimizer often does not converge for larger models. To combat this, we perform a variant of k-fold cross-validation in a sense that the model training is done on $K - 1$ folds of the train set and then the last fold is used for validation in early stopping. The resulting models are evaluated on the development set and the results are averaged. These results are used to rank the models. This process is repeated for every combination of model architecture and hyperparameters the grid search generates.

5. Results

Our experimentation showed interesting results. From the architecture and parameter grid, we chose 3 models for representation.

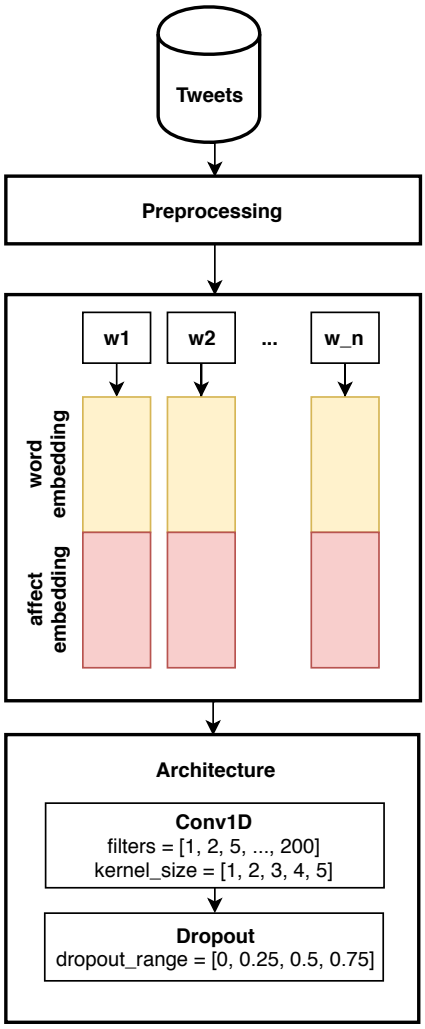


Figure 1: Model pipeline with an example architecture.

The first model, which we dub the *Minimal Convnet*, is as follows: `[Conv1D(2, 1), BatchNormalization, Dropout(0.25)]`.

The second *Small Convnet* is represented as: `[Conv1D(10, 3), BatchNormalization, Dropout(0.5)]`, and the third *Best Convnet* as: `[Conv1D(100, 3), BatchNormalization, Dropout(0.75)]`. The results are presented in Table 1. Surprisingly, the Minimal ConvNet with its two filters and kernel size 1 holds up admirably, scoring better results than the SVM baseline provided by the task organizers, but falling far behind the previous work. The Small ConvNet is our model of choice, which achieves solid performance while maintaining a very small architecture. This size of convolutional network seems to be optimal in our testing. Further increasing the model size only leads to worse performance and harder training. Although this model still falls behind the EiTAKA considerably, their implementation uses multiple word embeddings, multiple semantic lexicons, a much larger convolutional model with multiple channels combined with an XGBoost regressor into an ensemble. In the face of adversity, our Small ConvNet holds its own.

Table 1: Task 1.a results (emotion intensity regression), Pearson coefficient

	joy	anger	fear	sadness
EiTAKA	0.723	0.704	0.715	0.731
Best Convnet	0.634	0.605	0.611	0.627
Small Convnet	0.627	0.603	0.612	0.621
Minimal Convnet	0.587	0.573	0.569	0.591
SVM unigrams baseline	0.526	0.525	0.575	0.453
Random baseline	-0.018	0.024	-0.058	0.020

6. Conclusion

This work focused on finding the smallest viable model for the task of affect detection. To do this, we used the data provided in the first task of the SemEval 2018 competition. We created a basic framework for grid searching deep learning models and hyperparameters, and performed an extensive search in pursuit of the smallest model with acceptable performance.

While our models did not match the results of the top ranking teams, they did prove to be competitive in their own right.

In future work, it would be wise to find a richer sentiment lexicon than the affect lexicon we used. Despite our initial hopes, the semantic lexicon more often than not failed to offer values for given tokens since many of them were not a part of the relatively small vocabulary. None the less, it does not add a lot of overhead, and we believe that it would improve the model significantly.

References

François Chollet et al. 2015. Keras. https://keras.io.

Mohammed Jabreel and Antonio Moreno. 2018. Eitaka at semeval-2018 task 1: An ensemble of n-channels convnet and xgboost regressors for emotion analysis of tweets. *CoRR*, abs/1802.09233.

Mohammed Jabreel. 2016. The sentiment and emotion lexicons. http://sentiment.nrc.ca/lexicons-for-research/.

Olga Kolchyna, Tharsis T.P. Souza, Philip Treleaven, and Tomaso Aste. 2015. Twitter sentiment analysis: Lexicon method, machine learning method and their combination. abs/1507.00955.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Claim Strength Identification for Detecting Exaggerations in Science News

Leon Luttenberger, Kristijan Vulinović

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{leon.luttenberger, kristijan.vulinovic}@fer.hr

Abstract

The goal of this paper is finding a method for identifying claim strengths in science news as a step towards the detection of discrepancies between scientific publications and the media. Various models with different feature sets are implemented and compared with regards to performance. An analysis of the results is conducted in order to gain insight regarding the strengths and weaknesses of this system. Furthermore, the achieved results are on par with other related work.

1. Introduction

As information is able to spread extremely quickly across the globe, it is important to ensure its integrity. The results of scientific research are published in scientific journals, where they are subjected to a review process. Depending on the field, journal and paper, the paper is sent to multiple reviewers whose approval of the paper is necessary for the paper to be published. As a consequence of that, the findings which are reported in the paper must be accurate and therefore adhere to the standards of the journal and the scientific community in general.

However, as information spreads it gets reported again in news articles. It is important to understand that news articles are not subjected to the same standards as scientific journals. Moreover, while journals are meant to be read by people educated in the field in question, news articles are meant for everyone.

The following is an example of the difference in claim strength between a medical journal and a news article reporting the findings:

> *. . . bereavement is **associated** with reduced neutrophil bactericidal function. . .*
> *. . . intense grief **weakens** the body's immune system.*

The statement from the medical journal reports a correlation, while the news article implies a causation.

The way scientific developments are reported by the news has the potential to spread misinformation. This paper describes a method for identifying claim strengths in order to counteract the effect of misinterpretation. The mentioned approach requires three steps: claims extraction, claims pairing and claim strength identification. This paper focuses solely on the last step, while the other steps are left for future work.

2. Related Work

Claim strength identification is a seldom addressed topic in the field of Natural language processing (NLP). Most of the existing work focuses on detecting misinformation or rumors (Qazvinian et al., 2011). Those approaches often use sentiment and metadata such as hashtags, emoticons and retweets on Twitter besides word embeddings. The focus is usually on a single document or text.

Table 1: Categories of claim strength

Category	Description
0	No mentioned relationship
1	Statement of no relationship
2	Statement of correlation
3	Ambiguous statement of relationship
4	Conditional statement of causation
5	Statement of 'can'
6	Statement of causation

Unlike the described studies, Li et al. (2017) focus their work on claim strength identification across multiple documents. They trained a model to identify claim strengths in science articles published in medical science journals and news articles, with the ultimate goal of using the model to detect differences in claim strength between the two. The proposed model is a support vector machine classifier (SVM) with unigram bag-of-words features.

In this paper, we adopt the idea described in Li et al. (2017), but continue to experiment with other models and document representation techniques.

3. Data

3.1. Dataset

The data collected by Sumner et al. (2014) is publicly available online.[1] They identified press releases based on published studies with possible relevance to human health. The press releases were then associated with corresponding news stories. Each journal article, press release and news story was coded with a variety of labels.

Considering that the purpose of this paper is identifying claim strength based on text, only the statement of cause and claim strength label were used. The statement of cause refers to a single sentence or phrase extracted from the text which is used to identify the claim strength. The claim strength is coded into 7 categories with increasing strength

[1] https://figshare.com/articles/InSciOut/903704

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 75–78, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Table 2: Examples of claims with different claim strength categories

Category	Statement
0	we report the discovery and characterization of a unique core genome-encoded superantigen, providing new insights into the evolution of pathogenic S. aureus
1	the new treatment was found to be as effective as warfarin
2	Childhood obesity peaks between ages 7 and 11
3	prospectively rated childhood wellbeing has long-term beneficial links to adult functioning
4	CTLA-4 molecule plays a critical role in suppressing autoimmunity and maintaining immune homeostasis
5	losing a loved one can lead to parts of the immune system being suppressed
6	Premature babies risk mental health problems

Table 3: Category distributions before and after merging

Category	Original		Merged	
	Journal	News	Journal	News
0	0	2	-	-
1	39	43	39	43
2	257	117	337	191
3	80	74		
4	47	118	83	198
5	36	80		
6	395	469	395	469

of relationship, described in Table 1. Examples for each category can be found in Table 2.

3.2. Preprocessing

The original dataset contains 7 categories where category 6 (statement of causation) amounts to 49% of the examples while the other categories are relatively smaller.

With the intention of creating a more balanced dataset, the number of categories was reduced from 7 to 4. Firstly, category 0 (no statement of relationship) was removed as it was only represented by 2 examples from news stories and no journal articles. Categories 2 (correlation) and 3 (ambiguous relationship) were merged together due to their semantic similarity. Likewise, categories 4 (conditional causation) and 5 (statement of 'can') were also merged.

Table 3 shows the category distributions before and after preprocessing.

4. Experiment

4.1. Features

The simplest feature vector we used to encode documents was a term frequency bag-of-words representation. We did not remove stop words as it would remove some of the most relevant words for identifying claim strength, such as 'can', 'may', 'could', etc. Furthermore, seeing as those words are quite common across the corpus, we tried implementing a TF-DF vector in order to give higher weights to the most common words as opposed to a traditional TF-IDF vector. However, the method proved unsuccessful.

Additionally, we experimented with adding bigrams, stemming and part-of-speech (POS) tagging.

4.2. Models

Multiple different models were considered and tested. As a baseline, we used a simple majority class classifier. We used logistic regression and SVM with linear kernel. As the data originates from two diverse types of sources, the experiments include models trained on all the data without knowing anything about the source, models trained on science journals or news articles only and models trained on the whole dataset with an additional Boolean feature representing the source. Deep models were also considered but were left out because the dataset is not large enough to train them.

4.3. Results

All the hyperparameters for all the models are obtained using grid search, resulting in a slightly optimistic performance estimate. The models are then trained and tested using 10-fold cross validation, the results of which are shown in Table 4. All the models are tested separately on journals and news articles. The accuracies are visualized in Figure 1 for journals and Figure 2 for news. As it can be seen on the images, all the proposed models outperform the baseline, which is tested using Kruskal-Wallis test with 5% significance level. The resulting p-values are 2.28×10^{-7} for journals and 1.36×10^{-4} for news articles, which is way lower than the defined significance. Although the result displayed in Table 4 are not comparable with the ones in Li et al. (2017) because of the usage of different evaluation methods, it should be noted that the results are on par when using a fixed split for training and test set. However, as the fixed split approach is prone to have biased estimates, such results are left out of the paper.

[2]SVM with bigrams and stemming using two separate classi-

Table 4: Results

Model	Accuracy		Macro-average F1	
	Journal	**News**	**Journal**	**News**
Majority class	0.4628	0.5159	0.1576	0.1803
Logistic regression	0.7127	0.7620	0.6090	0.5984
SVM	0.6995	0.7408	**0.6431**	0.6008
SVM (tf-idf)	0.7021	0.7387	0.6414	0.5850
SVM (tf-df)	0.6176	0.7130	0.4131	0.5292
SVM (bigram)	0.7242	0.7618	0.6175	0.5972
SVM (bigram + stemming)	0.7359	**0.7636**	0.6163	**0.6145**
SVM (bigram + POS)	0.7075	0.7565	0.6063	0.5939
SVM (2 classifiers)[2]	0.7279	0.7581	0.5684	0.5854
SVM (source feature)[3]	**0.7488**	0.7588	0.6258	0.6078

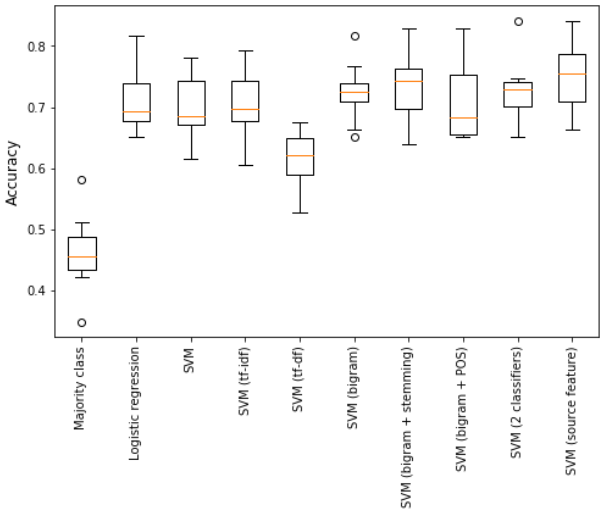


Figure 1: Box plot of model accuracies on journals

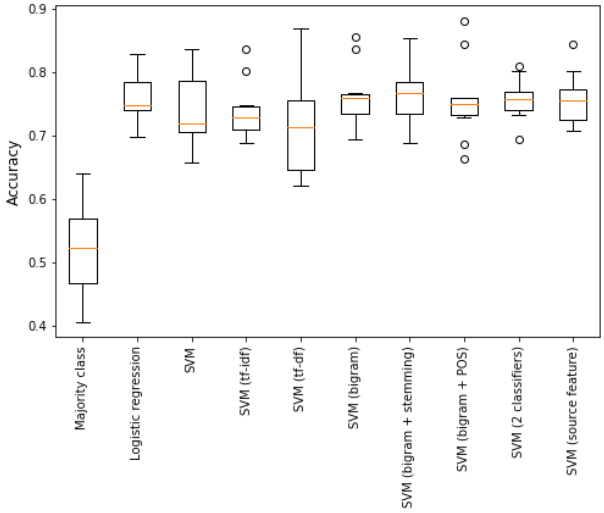


Figure 2: Box plot of model accuracies on news articles

Even though SVM received slightly higher macro-average F1 scores than logistic regression, it is proven not to be statistically significant on a 5% significance level using permutation testing. Because of that all the following test are done using SVM.

The previously proposed document frequency method is revealed to perform worse than the SVM with term frequency for journals, while not having a significant performance change for news articles by using permutation testing.

Leaving the baseline and document frequency classifiers out, Kruskal-Wallis test shows that there is no significant difference between all the other models with p-value over 0.4. Nevertheless, it is worth comparing the result for two separate classifiers, which show to not gain anything from the additional knowledge, mostly because of the even smaller training set available for each of the models. By checking the results for the model with the additional source feature it is concluded that the model suffers from overfitting, which is not strange keeping in mind that the number of features is about double the size of the dataset.

4.4. Error Analysis

Error analysis reveals that the models' most common error is differentiation between categories 2 (correlation) and 6 (causation). Figure 3 shows the confusion matrix based on the predictions of the SVM model with stemming and bigram features.

Additionally, we extracted the most influential bigrams for each of the categories which can be seen in Table 5.

fiers, one for journals and one for news articles

[3] SVM with bigrams and stemming using an additional feature representing the source

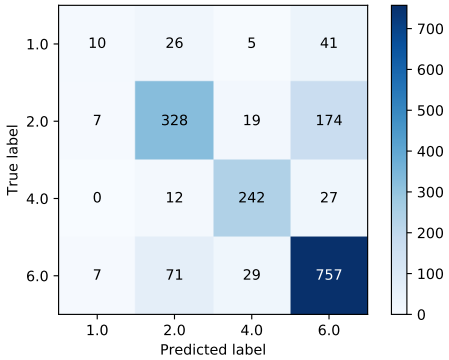


Figure 3: Confusion matrix for journal and news articles on 10-fold cross validation

Table 5: Most influential n-grams in favor of their respective categories, sorted by influence

Category	Words
1	no, not, tb, most, eczema
2	differences, linked, inappropriate, greater risk, linked to
4	can, could, may, appears, appears to
6	because, as likely, intelligence, as greater, is better

The bigrams were chosen based on the weights in the SVM model which correspond to the bigram mapping in the term-frequency vector. Intuitively, many of the bigrams seem appropriately chosen to represent their respective category. Category 2 is represented by 'linked', 'linked to' and 'greater risk', category 4 is represented by 'can', 'could' and 'might', and category 6 is represented by 'because' and 'as likely'. Unfortunately, there are also signs of the model overfitting. For instance, the model considers 'TB' (tuberculosis) and 'most' to be important indicators for category 1. Upon closer inspection, we found that the dataset contains many news articles referring to UK medical screenings missing most cases of TB.

5. Conclusion

The results of our experiments, much like others, suggest that claim strength identification is a challenging and still largely unexplored topic in NLP. Various models are compared in the paper, without any notable increase in performance.

Our analysis reveals that the main challenge is differentiating statements of correlation from statements of causation. Another issue is the small size of the dataset which renders the model prone to overfitting. However, despite the results leaving room for improvement, our model was able to successfully find relevant features for the task at hand.

References

Yingya Li, Jieke Zhang, and Bei Yu. 2017. An nlp analysis of exaggerated claims in science news. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*, pages 106–111.

Vahed Qazvinian, Emily Rosengren, Dragomir R. Radev, and Qiaozhu Mei. 2011. Rumor has it: Identifying misinformation in microblogs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1589–1599, Stroudsburg, PA, USA. Association for Computational Linguistics.

Petroc Sumner, Solveiga Vivian-Griffiths, Jacky Boivin, Andy Williams, Christos A Venetis, Aimée Davies, Jack Ogden, Leanne Whelan, Bethan Hughes, Bethan Dalton, Fred Boy, and Christopher D Chambers. 2014. The association between exaggeration in health related science news and academic press releases: retrospective observational study. *BMJ*, 349.

The Good, the Bad and the Ugly: Hate Speech Identification on Twitter

Domagoj Marić, Trpimir Zovak

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{domagoj.maric,trpimir.zovak}@fer.hr

Abstract

Hate speech is speech that is used to express hatred towards a targeted group, most commonly on the basis of race, religion or sexual orientation. On social media, it is usually manifested in the form of racist and sexist remarks intended to be derogatory, to humiliate, or to insult the members of the group. Using hand-engineered features used in related work, combined with some ideas of our own, we trained a multi-class classifier trying to distinguish between three categories of tweets: hate speech, offensive language and neither. We achieved competitive results in comparison to related work.

1. Introduction

When asking the question what hate speech is, there is no formal definition to answer it, but there is a consensus that it is speech that targets (usually disadvantaged) social groups in a potentially harmful manner (Jacobs and Potter, 2000; Walker, 1994). Hate speech is a common occurrence on the Internet, and can often result in severe threats to some individuals. Hate speech detection is critical for social networks to eliminate the problem, while still maintaining the right to freedom of speech (Waseem and Hovy, 2016).

We can also define some criteria to identify generally offensive tweets. The criteria is partially derived from the criteria presented in (Waseem and Hovy, 2016). A tweet is offensive if it:

- uses sexual, racial, religious or national slurs or defends sexism or xenophobia
- attacks or seeks to silence a minority
- misrepresents truth or seeks to distort views on a certain minority based on negative stereotypes
- promotes hate speech and/or violent crime or shows problematic hashtags (e.g. *"whitegenocide"*)

It is important to note that the presented definition does not include all instances of offensive language because people often use terms that are highly offensive to certain groups but in a different manner (Davidson et al., 2017). For example, words like *bi*ch* are commonly found in rap lyrics and homophobic slurs like *fa**ot* or *f*g* are commonly used by teenagers when playing video games. Most notably, some African Americans often use the *"N-word"* with a replacement of *"er"* with *"a"* in everyday language to indicate group solidarity (Stephens-Davidowitz, 2011; Warner and Hirschberg, 2012).

Previous work on the subject of hate speech detection has been able to make significant advances, but differentiating between offensive language and hate speech is still a problem (Davidson et al., 2017).

2. Related work

Most approaches in the field of hate speech detection yield decent results, but often lead to the misclassification of offensive tweets as hate speech. For example, bag-of-words approaches have a high recall, but have high rates of false positives (Kwok and Wang, 2013; Burnap and Williams, 2015). In their study, focusing primarily on racism, Kwok and Wang discover that 86% of the time the reason a tweet was categorized as racist was because it contained offensive words. This becomes a great challenge in hate speech detection because of the high rate of offensive and "curse" word usage on social media and the Internet in general (Wang et al., 2014). It is worth noting the presence of the opposite case – the occurence of some "special" tweets that do not contain offensive words or slurs usually connected to hate speech, but are clearly hateful towards a certain group (Davidson et al., 2017).

A relevant factor in detecting the difference between offensive language and hate speech lies in the context and subtle linguistic distinctions. Most notably, tweets containing the word *ni**er* are more likely to be classified as hate speech than those containing the word *ni**a* (Stephens-Davidowitz, 2011; Kwok and Wang, 2013), and the word *gay* is very ambiguous by itself because it heavily depends on the context (Wang et al., 2014).

There are also some approaches based on syntactic features, which rely on occurences of relevant words, like *kill* and *Jews* (Gitari et al., 2015), POS trigrams (Warner and Hirschberg, 2012), and the sytantic structures, like <intensity><user intent><hate target>, e.g. *"I f*cking hate ni**ers"* (Silva et al., 2016).

The basic ideas of our approach were partially derived from the work of Davidson et al. (2017), considering the efficient NLP approach and high recall. We used the same data set, but tended to further analyze the data to add new ideas and combine them with the ideas from related papers.

3. Data

3.1. Data set

We started with a *collection*[1] of 24783 tweets that were classified by CrowdFlower (CF) users as offensive, hate speech or neither (Davidson et al., 2017). Each tweet was coded by three or more people, and the intercoder-agreement provided by CF is 92%. Each entry contains

[1] http://github.com/t-davidson/hate-speech-and-offensive-language/tree/master/data

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 79–83, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

5 columns:

- *count* = number of CF users who coded each tweet
- *hate_speech, offensive_language* and *neither* = number of CF users who classified the tweet into the respective category
- *class* = class label for the majority of CF users

It was shown that the great majority of the entries was classified as offensive language (77 %), while the minority of 5 % were tweets classified as hate speech. In addition, 70,5 % of the tweets were classified by a unanimous vote.

3.2. Analysis

We collected a total of 51 commonly used offensive and hateful words (and word compounds) from online sources such as Wikipedia and *Urban Dictionary*[2]. For each word, we checked how many tweets in the data set contained the given word, and furthermore, how many of those tweets were classified as offensive or as hate speech (see Table 1).

Table 1: Occurrence analysis of some offensive words

word	total	offensive %	hate %
ni**er	317	48.58 %	51.10 %
ni**a	2188	90.04 %	9.83 %
white trash	94	37.23 %	59.57 %
fa**ot	539	53.43 %	46.38 %
f*g	837	56.75 %	42.65 %
bi*ch	10783	97.61 %	2.28 %
wetback	16	18.75 %	81.25 %

We also collected a total of 4278 commonly misspelled words from a machine-readable list on *Wikipedia*[3] and did the same for each of them. When we totaled the number of occurrences, it turned out that around 80 % of all misspelled words came from a tweet which was classified as offensive, and around 5 % of them from hate speech.

3.3. Ideas

The basic idea for our approach to tackle this task was to carry out the more or less standard Natural Language Processing (NLP) pipeline. The analysis shown in Section 3.2. has given us more ideas.

Firstly, we constructed two different files containing some of the 51 offensive words from our list. We chose only those words that have a big enough percentage (>10%) for offensive and hate speech occurrences. We also included those words that have either an extreme offensive or hate speech percentage, such as *bi*ch* or *wetback* (see Table 1). The files will contain the selected words sorted into 5 categories by type of word (racial, sexual, national/religious slurs, the "prostitute" category and general swear words) and 5 categories by hate proportions (percentage). These files will later be used for counting how many words from a certain category appear in a tweet. Secondly, we would use the list of commonly misspelled words and count how many of them appear in a tweet, similarly to offensive words.

Other tasks like counting Twitter mechanics (hashtags, mentions, emojis, etc.), counting words and syllables and calculating the Flesch reading-ease scores, were inspired by the work of Davidson et al. (2017). Our addition to these tasks is the uppercase ratio, which indicates how many letters were written as capital letters.

4. Natural Language Processing pipeline

For constructing features, we followed the extended standard NLP pipeline. The work done can be seen graphically summarized in Figure 1. For the majority of the NLP tasks we used the *nltk* Python package (Bird et al., 2009).

4.1. Cleaning tweets

Before doing the majority of the NLP tasks, we cleaned the tweets by doing the following:

- removing all hashtags, mentions, emojis and URLs
- removing quotation marks and ampersand characters
- replacing one or more consecutive special characters greater-than and less-than with *gt* and *lt*
- removing the apostrophe symbol in contractions of negated verbs (*shouldn't, can't, won't, ain't* ...)

4.2. Tokenization, stop word removal and stemming

We tokenized lowercased tweets by using the *nltk* Python package, and then removed all tokens that were stop words (e.g. *and, what, the, is* ...), by obtaining the set of stop words from the *nltk* corpus. In addition, we stemmed all tokens and verified that the relevant offensive words *ni**er, ni**a, fa**ot* and *f*g* were all stemmed differently.

4.3. Part-of-speech tagging and named-entity recognition

We also tokenized the tweets with their original case, because lowercasing them would have an influence on later POS tagging and named-entity recognition. The POS tagging has given each of the tokens one of the 37 available tags. In addition, for each POS-tagged tweet, we did the named-entity recognition to give tokens tags like *PERSON* and *LOCATION*. POS tags will later be used for n-gram features, while NE tags will be counted per tweet.

4.4. Collecting n-grams

We collected word, character and POS n-grams: unigrams, bigrams and trigrams for all of them, and extra character quadrigrams. N-gram features were weighted by their TF-IDF (*term frequency - inverse document frequency*) factors. Term frequency (TF) is defined as the ratio of how many times the n-gram appears in a tweet to how many n-grams are in the tweet in total. Inverse document frequency (IDF) is a measure of how common the n-gram is across the whole data set, obtained by dividing the total number of tweets by the number of tweets that contain the n-gram, and then logarithmically scaling the quotient.

[2] http://www.urbandictionary.com/

[3] http://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings

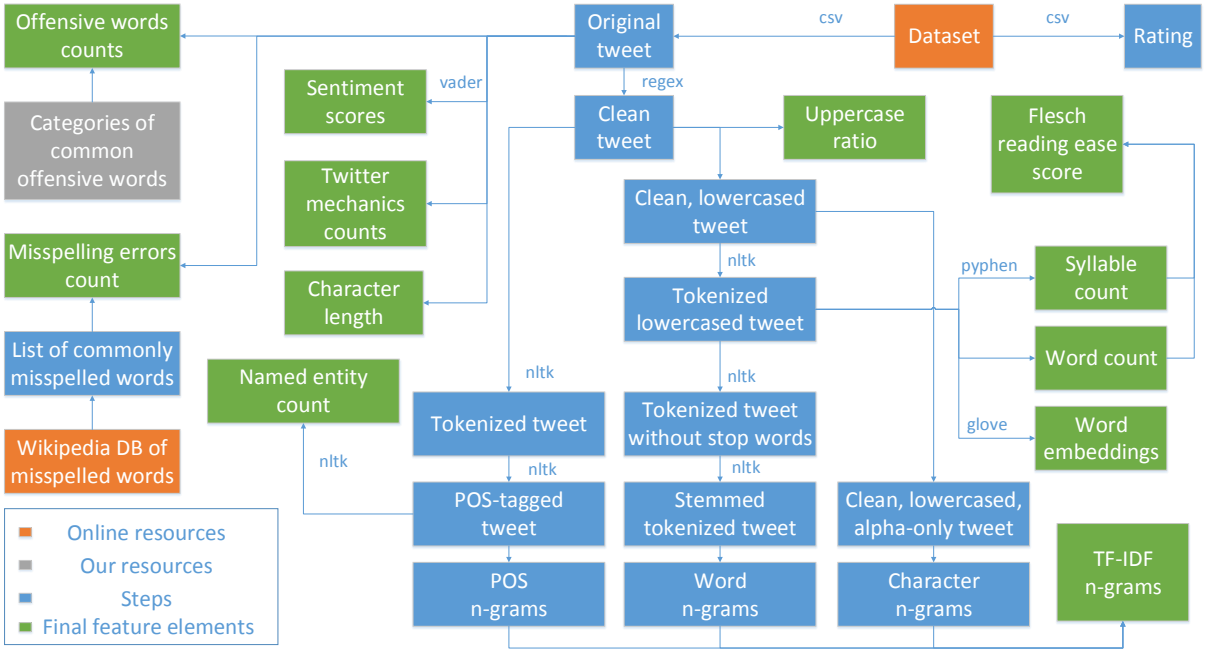


Figure 1: Graphical summary of done NLP tasks. The tasks without labels were done without using additional packages.

4.5. Flesch Reading Ease Score

We also calculated the Flesch reading ease score (FRES) of each tweet by using the formula:

$$FRES = 206.835 - 1.015(\frac{words}{sentences}) - 84.6(\frac{syllables}{words})$$

In the FRES test, higher scores indicate that the tweet is easier to read (Flesch and others, 1949). We obtained the number of syllables by using the *pyphen*[4] package, and set the number of sentences to 1 for all tweets because most tweets contain only one sentence, and it is a difficult task for a program to determine how many sentences there are, given the nature of tweets.

4.6. Counting occurrences on original tweets

In addition to the NLP pipeline, we did the following tasks on unmodified tweets:

- counting offensive word occurrences by category
- counting misspelling errors using the Wikipedia list
- counting hashtags, mentions, URLs and emojis

4.7. Sentiment scores

We used *VaderSentiment* (Gilbert, 2014) for assigning a sentiment score to each tweet. Four scores are assigned to each tweet: the positive, negative, neutral and compound score, which we will use as features.

4.8. Word embeddings

Using GloVe pre-trained word vectors (Pennington et al., 2014), each word of a tweet was transformed to a 200-dimensional vector. We summed all vectors of a tweet by their elements and used the final vector as a feature.

[4] `http://www.yabz.fr/`

5. Model

Before any preprocessing we split our data holding out 10% of it for testing while keeping the ratio of tweets classified as offensive, hate speech and neither (77/18/5). Each tweet was then transformed into a feature vector. Since 77% of the data is annotated as offensive, we created a baseline model which classifies every tweet as an offensive one. We tried a few combinations of features to see how they affect the performance of the model. For each set of features, we tried linear SVM and L2 Logistic Regression (Pedregosa et al., 2011), since they performed well in related papers. After choosing the model type and the set of features we performed the feature selection. First we tried using mutual information to keep our top N features. It improved performance of our model but it was slow and had no interaction with the model itself. We then tried removing all features whose importance was lower than the given threshold. This type of feature selection was performed over a set of C hyperparameters and a set of thresholds.

Each model was fine-tuned and evaluated by 5-fold cross validation using *StratifiedKFold* to preserve the percentage of samples for each class. We also tried recursive feature elimination, but the *SelectFromModel* approach (described above) gave the best results and it was also the fastest feature elimination method.

6. Results

Although there are 77% tweets classified as offensive and only 5% as hate speech, we decided to use average measures instead of weighted ones because we didn't want our model to "ignore" performance on the hate class. We can see the difference in our initial baseline, which has an F1 weighted score of 69%, but the average F1 score is only 29.33%. Using only word ngrams as features we accomplished decent results on the offensive and neither class,

Table 2: Importance comparison of different feature sets

	Linear SVM			L2 Logistic regression		
	Precision	Recall	F1	Precision	Recall	F1
Word n-grams	76.88	68.30	70.02	76.32	69.39	70.99
Character n-grams	77.12	68.01	70.09	75.80	68.60	70.81
Word + character n-grams	75.69	71.57	73.20	75.61	71.85	73.43
Word + character + POS n-grams	76.63	71.70	73.57	75.17	72.16	73.45
n-grams + offensive words and misspelling counters	78.45	72.50	74.66	79.33	71.91	74.52
All above + Twitter mechanics and length counters	77.29	72.92	74.67	78.51	73.00	75.11
All above + sentiment scores + word vectors	77.39	73.45	75.52	79.37	72.42	74.86

but results on the hate class are significantly worse. For that reason, we use the F1 macro measure to evaluate our model. Table 2 shows how each set of features affects the performance of our model. We can see that character and word n-grams are the most effective features. We found it odd that the F1 score of Logistic regression dropped when we added word embeddings, so we tried selecting features using Logistic regression with L1 regularization. We got improved performance with a train F1 score of 76.2%.

As our final model we chose L2 Logistic regression with all features. We trained it on the entire train set and tested it on the test set. Looking at the results in Table 3, we can see that the results on hate improved, but are still not close to the ones of the offensive and neither class. We dig deeper into the results to see why that may be.

Table 3: Final model (L2 Logistic regression)

	Precision	Recall	F1 score
neither	0.79	0.95	0.86
offensive	0.97	0.90	0.94
hate speech	0.40	0.57	0.47

We look at all the tweets that were misclassified. We decided to check how many of those tweets were annotated by a unanimous vote. Of all the tweets that were misclassified by our model, 73% of them were not classified by annotators in unison.

Tweets misclassified as hate speech seem to contain offensive words from Table 1, which have a high percentage of occurrence in the hate class, e.g. *"@ItsNotAdam fa**ot read my tweets after dat k" it wasn't even funny lol"*. While tweets may contain terms that can be considered racist and sexist, it is apparent that many Twitter users use this type of language in their everyday communication.

On the other hand, tweets which are misclassified as neither seem to contain uncommon offensive words or don't contain any offensive words at all, e.g. *"@TrapFuhrer: When her nudes trash you gotta draw clothes back on her and send em back"*, which indicates that our model has a problem with understanding the context.

Since we use the same data set as the one used in the work by Davidson and others, we compared our results to theirs. Because their model was not properly evaluated, we reimplemented their model and made a comparison using the same data split. As seen in Table 4, the reimplemented model has a train F1 score of 74.2% and a test F1 score of 72.3%, which is 2% and 3% lower than the performance of our model. We performed a two-tailed permutation test with a significance level of 5%, which showed that our model is statistically significantly better.

Table 4: Model performance comparison

	Baseline	Davidson et al. reimplemented	Our model
F1 average	29.33	72.3	**75.66**
F1 weighted	69	89	**90**

7. Conclusion

Although our model succeeds in identifying offensive tweets, it struggles with hate speech. The first problem we have is that the data consists of only 5% tweets annotated as hate speech. Only 1.3% of those 5% were annotated by a unanimous vote, which indicates that detecting and differentiating hate speech from offensive language is a difficult task even for a human. Human classifications for hate speech tend to reflect their own subjective biases, which leads to wrong annotations, and consequently, corrupted data. We can see that that is the case, since almost 75% of the misclassification occured on tweets which were not unanimously annotated. In addition, the model highly depends on the occurence of offensive terms and struggles with understanding the context. Therefore, we believe a good first step to better results would be to collect better data: expert and objective annotators, higher percentage of tweets annotated as hate speech, and more tweets classified as hate speech and offensive language, which do not contain offensive terms.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Pete Burnap and Matthew L Williams. 2015. Cyber hate speech on twitter: An application of machine classifi-

cation and statistical modeling for policy and decision making. *Policy & Internet*, 7(2):223–242.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515.

Rudolf Franz Flesch et al. 1949. Art of readable writing.

CJ Hutto Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf*.

Njagi Dennis Gitari, Zhang Zuping, Hanyurwimfura Damien, and Jun Long. 2015. A lexicon-based approach for hate speech detection. *International Journal of Multimedia and Ubiquitous Engineering*, 10(4):215–230.

James B Jacobs and Kimberly Potter. 2000. *Hate crimes: Criminal law and identity politics*. Oxford University Press.

Irene Kwok and Yuzhou Wang. 2013. Locate the hate: Detecting tweets against blacks. In *AAAI*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Leandro Araújo Silva, Mainack Mondal, Denzil Correa, Fabrício Benevenuto, and Ingmar Weber. 2016. Analyzing the targets of hate in online social media. In *ICWSM*, pages 687–690.

Seth Stephens-Davidowitz. 2011. The effects of racial animus on voting: Evidence using google search data. *Unpublished typescript. Google Scholar*.

Samuel Walker. 1994. Hate speech. *The History of an American Controversy, Lincoln (ua)*.

Wenbo Wang, Lu Chen, Krishnaprasad Thirunarayan, and Amit P Sheth. 2014. Cursing in english on twitter. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 415–425. ACM.

William Warner and Julia Hirschberg. 2012. Detecting hate speech on the world wide web. In *Proceedings of the Second Workshop on Language in Social Media*, pages 19–26. Association for Computational Linguistics.

Zeerak Waseem and Dirk Hovy. 2016. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL student research workshop*, pages 88–93.

Extraction of Drug-Drug Interactions

Erik Matošević, Ivo Žužul

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{erik.matosevic,ivo.zuzul}@fer.hr

Abstract

This paper describes problem of extracting drug-drug interaction from biomedical documents. Drug-drug interaction (DDI) is broadly described as a change in the effects of one drug by the presence of another drug. So it makes a sense to develop an information extraction system that is able to automatically identify and extract relevant information on DDIs. We approached this problem with developing information extraction system that preprocesses data, extracts features and trains on data using machine learning method based on linear SVM model.

1. Introduction

The drug-drug interaction (DDI) is a situation in which one drug changes its effects by the presence of another drug (e.g. *Acetazolamide reduces urinary excretion of quinidine and may enhance its effect.*). These interactions are mostly described in the medical journals, making them the most effective source for their detection. The detection of DDIs is an important research area, since these interactions can have a dangerous impact on patient safety and health care costs. That's why it is necessary to develop an information extraction system that is able to automatically identify and extract relevant information on DDIs, which would also benefit the pharmaceutical industry by reducing the time spent on reviewing the literature.

Our goal is to develop a system that can identify pharmaceutical substances from the biomedical literature and detect drug-drug interactions. The main task is divided on two subtasks, to enable evaluation of two different aspects of the system:

- Task 1: Recognition and classification of drug names
- Task 2: Extraction of drug-drug interactions

In the Task 1 there are four general types of drugs (DRUG, BRAND, GROUP, NO-HUMAN), and in Task 2 there are four types of DDIs that occur (ADVICE, EFFECT, MECHANISM, INT).

Dataset was obtained from DDI corpus which is a semantically annotated corpus of documents describing drug-drug interactions.

This paper is organized in sections: Section 2 describes related work, Section 3 describes dataset (DDI corpus), Section 4 describes proposed approach in solving the drug-drug interaction problem, Section 5 presents results and evaluation and Section 6 is a conclusion.

2. Related work

DDIExtraction 2011 (Segura-Bedmar et al., 2011) was the first event whose goal was to detect drug-drug interactions from biomedical texts. The participants needed to develop a system that can extract drug-drug interactions from the texts, but there was no classification of drug names in this task. Datasets used for this task were biomedical texts from a DrugBank database (Wishart et al., 2006). There were 10 participating teams and the best F1 score was 65.74%.

DDIExtraction 2013 (Segura-Bedmar et al., 2013) followed the event from the 2011, but the task had more additions to the previous one. Except for detecting drug-drug interactions, participants were also required to recognize and classify the types of the pharmocological substances, which divided the task into two subtasks, and while in DDIExtraction 2011 participants had to identify all possible pairs of interaction drugs, DDIExtraction 2013 included classification of drug-drug interactions into different types. The datasets were also expanded to include MedLine abstracts together with DrugBank database, so that system can work with different language styles from different texts. The participation exceeded to 14 teams and the best F1 score increased from 65.74% in DDIExtraction 2011 to 80%, while the best F1 score for the classification of drug types was 65.1%

3. Dataset Description

We use DDI corpus for training our Information Extraction (IE) system. The DDI corpus is a semantically annotated corpus of documents describing drug-drug interactions from the DrugBank database and MEDLINE abstracts on the subject of drug-drug interactions. The corpus has been first preprocessed to detect sentence boundaries and has been manually annotated with pharmacological substances (drugs) and interactions between them. It consists of 1017 texts:

- Drug Bank - contains 784 documents describing drug-drug interactions from the DrugBank database
- MedLine - contains 233 abstracts on the subject of drug-drug interactions.

For training, a dataset based on the publicly available portion of the DDI corpus is provided in XML. It contains two parts:

- Drug Bank - contains 572 documents describing drug-drug interactions from the DrugBank database

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 84–86, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

- MedLine - contains 142 abstracts on the subject of drug-drug interactions.

Each document consists of following elements:

- <document> element - root element of a document.
- <sentence> element - each sentence of a document is contained within a <sentence> element.
- <entity> - corresponds to all annotated pharmacological substances.
- <pair> - corresponds to all annotated drug-drug interactions.

4. Proposed approach

In this section, we describe the architecture of our system, which is divided into three stages:

- Preprocessing data
- Feature extraction
- Classification

4.1. Preprocessing

Strategy for preprocessing DDI corpus was to parse each document by its tags and save information from the tags above. Also, raw text was extracted from the <sentence> tag, so that training could be done on it. Each sentence was then normalized with the help of the NLTK library for natural language processing. From each sentence were removed most common English stopwords which do not matter too much, because we don't want these words taking up space in memory, or taking up valuable processing time. As said before, each <sentence> tag can have <entity> tags. In this case we are replacing words from <entity> tags in raw text with tokens: DRUG and OTHER_DRUG because we want to extract information about drug-drug interaction from other words in the sentence.

Next step was to define position of those other words in the sentence in relation to DRUG and OTHER_DRUG tokens. Therefore, words in the sentence that come before DRUG token were concatenated with *_bf* suffix, words between DRUG and OTHER_DRUG tokens were concatenated with *_be* suffix, and words that come after OTHER_DRUG token were concatenated with *_af* suffix. For example, if the sentence is *"Concomitant administration of Mefloquine and other related compounds (eg, quinine, quinidine and chloroquine) may produce electrocardiographic abnormalities and increase the risk of convulsions."*, then normalized sentence is *"Concomitant_bf administration_bf DRUG related_be compounds_be (eg_be ,_be OTHER_DRUG ,_af quinidine_af chloroquine)_af may_af produce_af electrocardiographic_af abnormalities_af increase_af risk_af convulsions_af ._af"*.

4.2. Feature extraction

We are using preprocessed data to convert it to training data and feature vectors, so we can train on it. From each normalized sentence are extracted unique tokens, with their suffixes. Also, bigrams are extracted from each sentence and joined to the set of unique tokens. After all unique tokens and bigrams are extracted, they are sorted and collected by frequency of occurence in all the documents. Feature vector is then made of number of occurences of unique tokens.

4.3. Classification

Model used for this drug-drug interaction classification problem is linear Support Vector Machine, because it performs particularly well in text classification tasks such as category assignment, detecting spam and sentiment analysis. We used linear SVC implementation from sklearn library which is based on libsvm. Kernel type is linear, and decision function is set to one-vs-rest shape.

5. Results and evaluation

Original dataset was divided into two parts: 80% of dataset is training set and 20% of dataset is test set. We got following results:

- Precision: 0.89
- Recall: 0.90
- F1-score: 0.90

6. Conclusion

In this paper we tackle the problem of drug-drug interaction with developed information extraction system that preprocesses data, extracts features and trains on DDI corpus. Sentences are then classified into positive or negative interactions between drugs that occur in text. Linear SVM model is used for training and classification. In the future, drug types occuring in sentence should be considered and differentiated so the system can classify inputs more precisely.

References

Isabel Segura-Bedmar *SemEval-2013 Task 9 : Extraction of Drug-Drug Interactions from Biomedical Texts (DDIExtraction 2013)*

Md. Faisal Mahbub Chowdhury *FBK-irst : A Multi-Phase Kernel Based Approach for Drug-Drug Interaction Detection and Classification that Exploits Linguistic Information*

Philippe Thomas *WBI-DDI: Drug-Drug Interaction Extraction using Majority Voting*

Richard Boyce *Using Natural Language Processing to Identify Pharmacokinetic DrugDrug Interactions Described in Drug Package Inserts*

Alexander L. Hayes *Predicting Drug-Drug Interactions from Text using NLP and Privileged Information*

Bethany Percha *Discovering New Drug-Drug Interactions by Text-Mining the Biomedical Literature*

Using Linguistic Metadata for Early Depression Detection in Social Media

Andrija Perušić, Denis Kustura, Ivan Matak

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{andrija.perusic, denis.kustura, ivan.matak}@fer.hr

Abstract

The goal of this paper is to use social media network as source for predicting early signs of depression. For that purpose we process posts written by *Reddit* users. The dataset consists of 892 labeled users. Along with *tf-idf* document vectorization we use hand-crafted features in our models. Purpose of those hand-crafted features is to detect and exploit subtle signs of depression in social media text which will in turn improve evaluation result. We use three classifiers: logistic regression classifier (LR), SVM classifier and voting classifier (consisting of LR and SVM combined). Further we evaluate all three models and perform ERDE measure evaluation on all three of them. Also we present those results in the paper and show that hand-crafted features have a positive impact on evaluation.

1. Introduction

Depression negatively affects how we feel, the way we think and the way we act. It causes the feeling of sadness and a loss of interest in any activity. By WHO statistics, it affects more than 300 million people with an increase of depression of more than 18% between 2005 and 2015.

This worrying statistics impose question how to detect depression as early as possible before any significant damage is caused. One of possible, and logical, answers is to examine the area where most of the people nowadays spend lot of their time which is social media. Social media is area where human social behavior comes to the fore and where people often express their feelings and problems so that makes it ideal environment for depression detection. A lot of work has been done in the area of early depression detection through NLP and machine learning, both deep learning approach and basic model approach with feature engineering. However, even though social media is a rich source of data, detecting depression is still hard because signs of depression are subtle.

In this paper, we concentrate on finding those subtle signs of depression, by modeling existing knowledge about language use of depressed individuals into set of features for our classification models. Models are trained and tested on a dataset which consists of labeled *Reddit* posts. In following sections we present detailed explanation of dataset, additional features along with a description and evaluation of models that use them.

2. Related work

A lot of work has been done in development of quality tools for early depression detection and there are constant improvements in this area. Also a lot of psychology studies have been conducted to further explore language use of depressed individuals.

We use results from several of these studies to hand-craft informative features. In the psychology study of (Rude et al., 2004) essays written by college students were examined and it was shown that elevated use of first-person pronouns (especially word "I") is connected to depression. In psychological research of (Al-Mosaiwi and Johnstone, 2018) over 63 different Internet forums absolutist words were used as indicators for depression and we use this list of words.

In area of developing tools and methods for early depression detection (Losada and Crestani, 2016) presented a new dataset of *Reddit* posts collected over long period of time which stores information about depression evolution. We train and test our models on this dataset. They also introduced new evaluation metrics, *ERDE* measure, which is designed especially for detecting early signs of depression. We used this measure for evaluation.

In work of (Trotzek et al., 2017) different models (BOW models RNN, LSTM), were employed for depression detection. Also in their work they used hand-crafted features beside standard document vectorization methods to improve evaluation results. Another work that concentrates on feature engineering is (Stankevich et al., 2018) where they used stylometric and morphology features with standard document vectorization methods and showed that additional features have an impact on improving evaluation scores.

In this paper, we will examine effectiveness of *tf-idf* document vectorization with our additional features that were made by exploiting all of previous knowledge of psychological studies on depressed people and by using *Empath* (Fast et al., 2016), text analysis tool that can generate and validate new lexical categories on demand.

3. Dataset

For training and testing we used dataset that consists of *Reddit* posts collected by (Losada and Crestani, 2016) from 892 users. It consists of 137 depressed users and 755 non-depressed users and it is split into 486 users for training set (83 depressed and 403 non-depressed) and 406 users for test set (54 depressed and 352 non-depressed).

Dataset is created as sequence of XML files, one file per user. Each XML file stores the sequence of the users' submissions (one entry per submission). For most active users there are up to 2000 submissions (1000 posts and 1000 comments) and those submissions include submission to any *subreddit* category. Every submission in XML file is represented with submission's title, text and date. Also

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 87–91, University of Zagreb,
Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Table 1: Most informative semantic categories.

Category	Average		Difference
	Positive	Negative	
friends	0.00846	0.00505	**0.00341**
positive_emotion	0.00899	0.00597	**0.00302**
negative_emotion	0.01224	0.00924	**0.00300**
love	0.00521	0.00259	0.00262
nervousness	0.00419	0.00159	0.00260
pain	0.00594	0.00334	0.00260
shame	0.00489	0.00236	0.00253
optimism	0.00652	0.00412	0.00240
sadness	0.00361	0.00145	0.00216
speaking	0.00910	0.00696	0.00214

posts in every XML file are ordered chronologically for every user. This dataset is specific because it does not only allows us to find differences in language use between depressed and non-depressed users but it also allows us to see evolution of language use of depressed users during a longer period of time.

4. Baseline experiments

The *bag-of-words* approach was taken for our NLP pipeline. For each user, all post were read sequentially and from each post the contents of tags *TITLE* and *TEXT* were concatenated. Other tags do not have information suitable for this task. The resulting document for each user was a concatenated string containing all their submission content. The data is raw and often titles and text contain noise (links, titles from a different source, etc.). We used multiple regular expressions to extract and remove this noise from data so that our final concatenated document contains only text written by a user. Finally, each cleaned document was then tokenized and stemmed with a *Snowball stemmer*. As the last step we made a *tf-idf* vectorized representation of cleaned, tokenized and stemmed document.

Classification was done by logistic regression (LR) with L1 regularization and SVM with linear kernel to asses the difference in performance. This dataset contains 755 users in control group and only 137 users labeled as depressed so it is extremely unbalanced classification problem. To address this issue and avoid building a trivial classifier that classifies everything as not depressed we adjusted misclassification costs for each class. Class weight is inversely proportional to class frequencies in the input data and is calculated by (1)

$$w_i = n_s/(n_c * c_i) \tag{1}$$

where n_s is number of samples in a dataset, n_c number of classes and c_i is number of samples labeled as class i in the dataset.

Both baseline models where optimized for optimum performance with standard grid search of two hyperparameters: k - number of k-best features to retain, calculated by a χ^2 distribution and C - a regularization factor. Each combination was evaluated by 5-fold cross validation on training set optimizing for F1 score of minority class. Cross validation process gave following optimal values $k = 1500$ for both models and $C = 16$ for Logistic Regression and $C = 8$ for SVM.

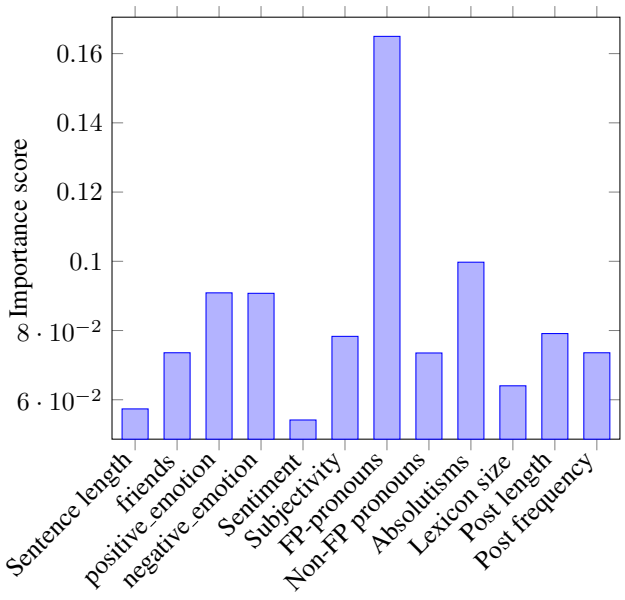


Figure 1: Feature importances.

5. Linguistic metadata features

Next step we took was attempt at modeling the existing knowledge about language use of depressed individuals as extra features that we later added to our baseline models. In addition to those features, we added several features that were based on our intuition about this classification problem. Total of 12 new features were created. In the next part of this chapter we list and explain all of new features.

Semantic feature set: In (Losada et al., 2017), analysis of emotions and topic signals with lexicons like LIWC (Linguistic Inquiry and Word Count) is advised, as it is deemed beneficial for depression detection. For out semantic features, as we call them, we used *Empath*. *Empath* is a lexicon mined from modern text on the web with 196 built-in, pre-validated lexical categories(Fast et al., 2016). Its categories are highly correlated with similar categories in LIWC. The reason we used Empath over LIWC is because it is open-source, has more potentially useful categories and is overall a bigger lexicon.

For a given text, Empath returns the count of words correlated to each of its lexical categories, normalized over total number of words. To extract categories useful for our problem, we summarized the normalized category scores for all positive and negative training examples and averaged the results separately. We then calculated the difference between the two averages for each category and picked the top 10 categories. To further optimize feature selection, we took only categories with difference score greater than difference mean average of top 10

categories. After optimization we were left with three lexical categories that show biggest distinction in language use between depressed and non-depressed individuals. Top categories are shown in Table 1.

Pronoun frequency: There is confirmed correlation of elevated use of first person pronouns and depression (Karmen et al., 2015). In contrast, there is also correlated reduced usage of non-first person pronouns. This reflects the phenomenon that depressed individuals are often focused on themselves and much less on others. We constructed two features. One that represents first-person pronoun frequency and the other for non-first person pronoun frequency for each user. Frequency is calculated as total number of pronouns divided by word count of most frequent word in a document.

Absolutisms frequency: Absolutist thinking is considered a cognitive distortion by most cognitive therapies for anxiety and depression and they track the severity of affective disorder more faithfully than negative emotion words (Al-Mosaiwi and Johnstone, 2018). We use a list of absolutist words from this paper, augmented by additional words we added, to devise a feature in a same way as pronoun frequency - number of absolutist words divided by word count of most frequent word in a document.

Sentiment and subjectivity: Negative sentiment is also often sign of depression or anxiety (Al-Mosaiwi and Johnstone, 2018). We use a learned model from *TextBlob* library to extract sentiment and subjectivity information. Model returns polarity $[-1.0, 1.0]$ and subjectivity $[0.0, 1.0]$ score for each sentence. We use this to devise two features as polarity (sentiment) and subjectivity averaged across sentences of a document.

Lexicon size: This feature is calculated as number of different words divided by total number of words in a document. This feature is simplifed version of a set of features proposed in (Trotzek et al., 2017) where they use measures for text readability (namely Gunning Fog Index, Flesch Reading Ease, Linsear Write Formula, New Dale-Chall Readability). Those measures store information about language complexity for an individual, which can be connected to depression.

Other: We created 3 additional metadata features on intuitive presumption they hold information that can improve our models. Their impact was tested in evaluation process. These features are average sentence length, average post length and post frequency. Average post and sentence length are calculated on a word basis and post frequency is calculated as number of minutes between first and last post divided by number of posts in a dataset for a given user.

All of the additional features were standardized by removing the mean and scaling to unit variance. The idea was to make a more sophisticated model by adding these new features to existing 1500 *tf-idf* baseline features and evaluate their performance.

6. Evaluation

As a first test we did not optimize feature selection. All new features were added to both baseline models and only regularization factor was optimized. It is important to note that all hyper-parameter optimization in our tests was done by 5-fold cross validation on a training set, optimizing for F1 score of minority class. This way the unbalanced nature of dataset was addressed. As we predicted, models had poor overall performance because there was a lot of noise in the dataset and subsequently, our features. For logistic regression the cross-validation score was worse by more that 3% and for SVM 0.02%. Nevertheless, we tested these models with optimized regularization factor on a test set and got the result shown in Table 2.

It should be noted that, even though the overall F1 score is worse than baseline, for LR with all features we got the biggest recall on test set. This is arguably very important trait because it is better to have false positives that can be later labeled as non depressed that completely miss depressed cases.

The next step was feature selection. For 12 new features, there are 2^{12} possible combinations and it was too resource intensive to cross-validate all combinations as a hyper-parameter of a model. Thus we used tree-based feature selection method. Tree-based estimators can be used to compute feature importances, which in turn can be used to discard irrelevant features. We used extremely randomized trees classifier with 500 trees in the forest and fitted it with only the new features (without *tf-idf* features) and labels of the training set. Calculated feature importances are shown in Figure 1. The threshold for labeling feature as important was mean average of all importances and this approach selected following features, sorted by level of importance: *FP pronouns, absolutisms, positive_emotion* and *negative_emotion*.

We added the optimal feature subset to baseline features and again cross-validated the regularization factor. We ended up with following values: $C_{LR} = 22$, $C_{SVM} = 10$. In case of logistic regression cross validation score showed 1% improvement over baseline, and for SVM, there was a slight improvement of 0.1%. Results on a test set show improvement over baseline models with roughly 2% and 1.5% rise in precision for LR and SVM respectively while retaining recall score.

With an effort to further improve we combined two models into soft voting ensemble which predicts the class label based on the argmax of the sums of the predicted probabilities of each model. Ensemble gave the best best precision score on a test set.

We have also conducted significance tests, where we used paired t-test on F1 score, at 95% of significance level and results showed that our SVM with optimum feature subset outperforms both baselines and our LR model. However, it did not outperform our voting classifier. On other side, our LR model failed to outperform any of other models on 95% significance level. Even though our voting classifier showed improvement on evaluation over all models presented in the work it has not succeeded to outperform nor baselines nor SVM on 95% significance level.

Table 2: Model results on test set.

Model	Precision		Recall		F1	
	Minority	Overall	Minority	Overall	Minority	Overall
Baseline						
Logistic Regression	0.507	0.893	0.704	0.870	0.589	0.878
SVM	0.520	0.897	0.722	0.874	0.605	0.883
All additional features						
Logistic Regression	0.462	0.895	**0.778**	0.850	0.579	0.865
SVM	0.524	0.884	0.611	0.874	0.564	0.878
Optimized feature subset						
Logistic Regression	0.528	0.896	0.704	0.877	0.603	0.884
SVM	0.534	0.899	0.722	0.879	0.614	0.887
Soft voting (LR + SVM)	**0.607**	0.898	0.630	0.897	0.618	**0.897**

Table 3: Comparison of ERDE results for our final models. * - (Losada and Crestani, 2016)

Model	P	R	F1	$ERDE_5$	$ERDE_{50}$
LR	0.39	0.78	0.52	11.82%	7.50%
SVM	0.45	0.69	**0.54**	11.21%	7.14%
Soft voting	0.49	0.76	**0.59**	10.95%	6.83%
LR*	0.40	0.78	0.53	6.00%	5.30%

6.1. ERDE(Early risk detection error) measure

In addition to standard evaluation measures, which we used to assess the system's output with respect to golden truth judgments, we employ the ERDE (early risk detection error) metric proposed in (Losada and Crestani, 2016) and the goal is to minimize its value. This is a time-aware measure which detects system's performance considering how early it gives its predictions and how correct they are. The decision delay is measured by counting the number of distinct textual items seen before giving the final output. Considering detection of risk cases is the main problem, the ERDE measure associates greater costs to true positives and false negatives than to true negatives and false positives.

In domains like this one late detection of positive cases has severe consequences, therefore ERDE scoring function multiplies the cost of true positives by a latency factor. This implements the idea that late detection of risk cases is equivalent to not detecting them at all. The latency factor is increasing with the number of seen items. To control the delay at which the cost grows more quickly, ERDE measure is parameterised by parameter o. To test out system we used two versions of the proposed metric: $ERDE_5$ and $ERDE_{50}$ which were used in previous work. It should be noted that no latency cost is introduced for true negatives because non-risk cases would not demand early intervention.

In order to process the stream of texts written by each user with our developed model, we utilized a dynamic method proposed in (Losada and Crestani, 2016) which incrementally builds a representation of each user, taking one of his posts at a time, and only makes a positive decision if our classifier outputs a confidence value above an arbitrarily initialized threshold of 0.5. If the stream of user's text posts gets exhausted the method outputs a negative decision.

The results of our ERDE tests compared to ERDE results of (Losada and Crestani, 2016) are shown in Table 3. We should state that our results are directly comparable only with this original results because other results are from CLEF 2017[1] where researchers had to process the documents in weekly chunks and it was not possible to submit predictions before processing a complete chunk.

Comparing our results to those in (Losada and Crestani, 2016) it can be seen that our model yields slightly worse ERDE scores but a better F1 score. We infer the better ERDE score is a result of higher recall and lower cost for false positives in the ERDE scoring function. The overall system performance should be observed as a joint of ERDE and F1 measures.

7. Conclusion

In this paper, we explore and extract various possible features hidden in one's social media posts in an attempt to improve results on the task of early detection of depression risk cases. We create several models, using several word-, sentence- and document level features, which we train and test on CLEF 2017 eRisk pilot task dataset. Proposed models improved performance after introducing the crafted features.

In future work, we would like to further explore possible features underlying a textual document which could lead to depression detection. Also, incorporating link analysis for many hyperlinks we removed from the dataset using regular expressions could be useful but should be carefully approached because of possible digression from original data.

[1] http://clef2017.clef-initiative.eu/

References

Mohammed Al-Mosaiwi and Tom Johnstone. 2018. In an absolute state: Elevated use of absolutist words is a marker specific to anxiety, depression, and suicidal ideation. *Clinical Psychological Science*, pages 1–14, January.

Adrian Benton, Margaret Mitchell, and Dirk Hovy. 2017. Multitask learning for mental health conditions with limited social media data. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 152–162. Association for Computational Linguistics.

Ethan Fast, Binbin Chen, and Michael S. Bernstein. 2016. Empath: Understanding topic signals in large-scale text. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 4647–4657, New York, NY, USA. ACM.

Christian Karmen, Robert C. Hsiung, and Thomas Wetter. 2015. Screening internet forum participants for depression symptoms by assembling and enhancing multiple nlp methods. *Computer Methods and Programs in Biomedicine*, 120(1):27 – 36.

D. Losada and F. Crestani. 2016. A test collection for research on depression and language use. In *Proc. of Experimental IR Meets Multilinguality, Multimodality, and Interaction, 7th International Conference of the CLEF Association, CLEF 2016*, pages 28–39, Evora, Portugal, September.

David E. Losada, Fabio Crestani, and Javier Parapar. 2017. erisk 2017: Clef lab on early risk prediction on the internet: Experimental foundations. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, pages 346–360, Cham. Springer International Publishing.

Stephanie Rude, Eva-Maria Gortner, and James Pennebaker. 2004. Language use of depressed and depression-vulnerable college students. *Cognition and Emotion*, 18(8):1121–1133.

Maxim Stankevich, Vadim Isakov, Dmitry Devyatkin, and Ivan Smirnov. 2018. Feature engineering for depression detection in social media. In *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM,*, pages 426–431, Funchal, Madeira, Portugal. INSTICC, SciTePress.

Marcel Trotzek, Sven Koitka, and Christoph M. Friedrich. 2017. Linguistic metadata augmented classifiers at the clef 2017 task for early detection of depression. In *Working Notes of CLEF 2017 - Conference and Labs of the Evaluation Forum*, volume CEUR-WS 1866, Dublin, Ireland, September.

Andrew Yates, Arman Cohan, and Nazli Goharian. 2017. Depression and self-harm risk assessment in online forums. *CoRR*, abs/1709.01848, October.

Hate Speech Identification Using Active Learning

Mateo Stjepanović, Mislav Žabčić, Filip Tolić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{mateo.stjepanovic, mislav.zabcic, filip.tolic}@fer.hr

Abstract

In this paper, we tackle the problem of detecting hate speech and finding a difference between hate speech and offensive language. We know that some machine learning approaches got good results when distinguishing the hate speech from offensive language. Problem with almost all machine learning approaches is that we need a lot of data to train the model properly. The main idea of this work is not to fix the results of previous machine learning methods, or to outperform them, but to get similar results with much less data on which model can learn. Our results show that using even baseline semi-supervised active learning model can reduce the data needed to train the model on the same level.

1. Introduction

The impact that social media have on our daily lives has kept growing for the last 15 years. With the appearances of social media like Twitter, Facebook, Tumblr and many others, interactions between the users and the ability to express their opinions have never been easier. The rise in the use of hate speech, incited by the evolution of social media, was followed by the need to detect it. Facebook CEO, Mark Zuckerberg, predicts that in a 5 to 10 year period they will have the AI tools necessary to build a successful hate speech detection machine. What makes the detection of hate speech such a difficult task? To address that question we have to define hate speech. We define hate speech as communication that expresses hate towards a certain group of people based on their race, ethnicity, national origin, gender, religion, sexual orientation, etc. It often includes the use of offensive language or cuss words. It is important to differ offensive language from hate speech because people often use offensive language in their everyday language. Most rap songs, which people often cite, are filled with offensive words such as b*tch, n*gga, wh*re, etc. While communicating with their closest friends, people often use cuss words as a joke with no intention of hurting the other person. Because of situations like these, we need to build a classifier that recognizes the difference between hate speech and offensive language.

We use the dataset annotated by CrowdFlower workers. The workers were asked to annotate 25000 tweets into 3 categories: hate speech (class 0), offensive language (class 1) and speech that contains neither hate speech nor offensive language (class 2). A clear imbalance in the dataset is shown in a histogram displayed in Figure 1. Davidson et al. (2017) used logistic regression to detect hate speech. Even though their model was rather successful, they still struggled with separating hate speech from offensive language. In this paper, we explore the use of active learning to deal with the problems of separating offensive language from hate speech and solving the imbalance of the used dataset.

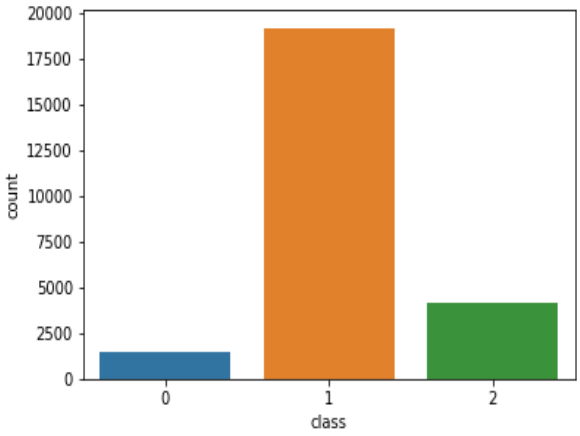


Figure 1: Histogram shows the imbalanced dataset. Tweets from dataset were extracted using *hate* words from Hatebase but not many of them contained hate speech.

2. Related work

Hate speech is a big, unresolved problem in the modern-day society. Even companies as big as Facebook are still struggling with it. It is important to separate hate speech from offensive language, even so, because of new laws against hate speech (Davidson et al., 2017). As said in Davidson et al. (2017) offensive language is mostly used in everyday life and often does not do any harm. Authors in that same paper tried few variants of models and decided that Logistic Regression with L2 regularization is the best model right now. While this problem is yet to be solved there is no discovered set of features that surely works. In Davidson et al. (2017) they used several features to capture information about syntactic and semantic structures. Porter stammer is used to create unigram, bigram and trigram features and then used TF-IDF to put weights to them. Using NLTK library they constructed Part of Speech tags as features. They showed that their model is working great, and got pretty high official metrics (precision, recall and F1 score). In the introduction to their paper, they stated that lexical struc-

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 92–95, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Tweets before cleaning	Lmao RT @MoeMartin44 The in soles in Reebok Classics can't even handle diabetic weight Rick Ross holds
Array of words	lmao, sole, reebok, classic, can't, even, handl, diabet, weight, rick, ross, hold

Table 1: First row shows the original tweet post from the dataset, the second row shows cleaned, tokenized and stemmed tweet post

tures often mislead model and produces false positives on hate speech. (Davidson et al., 2017)
Waseem et al. (2017) stated and gave an example of one reason why today we cannot get relevant scores or datasets. As the stated problem is that there is no consensus on what falls under what category. So they proposed a new separation method. It consists out of two questions (Waseem et al., 2017):

- *Is the language directed towards a specific individual or entity or is it directed towards a generalized group?*
- *Is the abusive content explicit or implicit?*

Answering those questions we can differ between four main classes:

- *Directed Implicit*
- *Generalized Implicit*
- *Directed Explicit*
- *Generalized Explicit*

Chen and Lin (2006) shows that one can implement feature selection methods into SVM itself. This could possibly enable great future work in trying to achieve an end-to-end solution for the hate speech identification problem, as we encountered the problem of a high number of features, and high CPU and RAM requirement as result of that. In our paper, we used baseline features because we did not want to focus ourselves on feature selection. The approach proposed by Chen and Lin (2006) gives us an idea of how to improve our model later on.
The problem of sufficient data size and the time needed to manually label the data has always been present in machine learning. That is why active learning as the semi-supervised approach is developed. Luo et al. (2005) proposed a new active learning algorithm based on multi-class support vector machines. They showed that there is a way to make SVM work with probabilities and gave us a proper active learning algorithm. Even though we did not implement this model, the plan is to implement it in the future and test it on state-of-the-art models. Their algorithm works as follows:

- 1. Start with an initial training set and an unclassified set.
- 2. A multi-class support vector machine is built using the current training set.
- 3. Compute the probabilistic outputs of the classification results for each data on the unclassified set. Suppose the class with the highest probability is a and the class with second highest probability is b. Record the value of P(a) and P(b) for each unclassified data.
- 4. Remove the data from the unclassified set that has the smallest difference in probabilities between them $(P(a) - P(b))$ for the two highest probability classes, obtain the correct label from human experts and add the labeled data to the current training set.
- 5. Go to 2

Yang et al. (2009) introduced their own active learning model based on multi-class SVMs. It is a little different from the one before in a way that they calculate probability on each data in the unlabeled pool. Considering that, their model is computationally expensive if unlabeled pool in large size (Yang et al., 2009). Their experiment also shows that they outperform state-of-the-art models, which can only be a sign to continue further on our work to create an end-to-end solution for hate speech identification.

3. Preparing data

We used the dataset that was introduced in Davidson et al. (2017). They prepared the data by annotating the twitter posts with three available classes: 0 meaning that the tweet contains hate speech, 1 meaning it contains offensive language, 2 meaning it contains neither of the previous two. The tweet was classified by the majority of the votes from the annotators from CrowdFlower. The minimum number of annotators for each tweet was three. As seen in Figure 1 data is unevenly distributed. That has a big influence on learning outcomes. We used traditional NLP methods to prepare the data.
First, we had to clear the text from all of the usual twitter "stuff", like hashtag or mentions. To remove these certain things, we use regular expressions to automate the process. Together with hashtags and mentions, we removed URL, times, dates.
After clearing the text, we switched it to lower case and converted it to tokens. These tokens were stemmed using Porter stemmer (Por, 2009).
Using *NLTK* library (NLT, 2009) we removed all classical stop words in the English language. After tokenization, 50 of the most frequent words are removed (same as removing dataset specific stopwords), that allowed us to lower the number of our features. That same size was one of the biggest problems in our work. After testing the removal of said words, we got considerably better results on our baseline models so we used the same approach in our active learning based model.
The last thing we had to do is to turn these tokens into a vector. We used TF-IDF vectorizer, from the *sklearn* library (Pedregosa et al., 2011), to extract features from our text. After using all these methods, we were ready to train our baseline and our active learning models

Table 1 shows the output of our data preparation. After text cleaning, tokenization and so on, we got an array of

Table 2: Table shows how precision, recall and F1 score behaves on different iterations.

Iteration	0			40			60			100		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Class 0	0.11	0.04	0.06	0.23	0.13	0.17	0.35	0.20	0.25	0.37	0.19	0.25
Class 1	0.80	0.91	0.85	0.82	0.90	0.86	0.82	0.91	0.86	0.82	0.91	0.86
Class 2	0.47	0.30	0.37	0.52	0.37	0.43	0.53	0.36	0.43	0.51	0.36	0.42

words that represents the input tweet from the dataset.

4. Model Overview

As stated above, one of the biggest problems is that we need a lot of annotated data to train the model properly. This problem presents itself in the hate speech identification as there is no relevant or good enough database. Database introduced in Davidson et al. (2017) suffers from the lack of equal distribution of data across all three classes. To sidestep that problem, we decided to go with a semi-supervised model. A model whose training is based on, so-called, active learning.

While this is just a brief overview and some kind of prototype to show us if active learning is a good approach, we introduced just one of the baseline models for active learning. Label Spreading and Label Propagation are the only two semi-supervised model that one can find in the *sklearn* library (Pedregosa et al., 2011) and we decided to go with the latter one.

The main idea of active learning is that in the first step we take a small subset out of the training set and train our model on it. In the next step, we need to calculate the probabilities of a good estimation of the model for the remaining training set. Given these probabilities, we then choose a subset of that data that has the lowest probability of estimation. After that model is trained on the calculated set. We repeat the previous steps until we either reach the maximum number of iterations or gain satisfactory results. The active learning algorithm can be summed in five steps:

- *1. Select m size subset from the training set*
- *2. Train label spreading model on given subset*
- *3. Predict probabilities on the remaining training set*
- *4. Select n worst probabilities and use that data to expand the initial training set*
- *5. Go to 2.*

In our work, we set the maximum number of iterations and training subset size to 100. These numbers were chosen to better capture the peak point of the algorithm. If we chose too big of a number, we could not accurately determine when did the model reach optimal results. On the other hand, if we chose too small of a number, training a model on a such a small subset would not make much sense. With these hyperparameters, we got the desired results after half of the maximum iterations, and we kept running the other half to see how the model will behave.

Table 2 show how are metric improved through time. In table 2 we can see that the optimal results are achieved after only 60 iterations. That is about 6000 labeled data on which we train our model.

In next section, it is shown that our model, based on active learning, even though it is baseline model, has a lot of potential for future work.

5. Experiments

We compare our active learning model to the model created by Davidson et al. (2017) and two different baseline models using the same features as our model. One of those models is a logistic regression and the other is an SVM. SVM and Logistic regression were evaluated using nested cross-validation while our active learning model was evaluated on a regular cross-validation. For SVM and Logistic regression *C* and *gamma* were optimized as hyperparameters. Nested cross-validation used 10 outer and 5 inner folds. For our results, we use those that gave us the highest F1 score for hate speech class. The results are shown in Table 4. Davidson et al. (2017) model was trained using 10000 TF-IDF features including POS tags and some general information features as tweet length, number of words, etc. Our 3 models used only 1000 TF-IDF features. We can see that because of the lack of features the F1 scores for hate class fall significantly compared to Davidson et al. (2017) but they are consistent for all 3 models. We assume the reason for this is that the sklearn implementation of TF-IDF vectorizer, used in this paper, takes only top used ngrams from the dataset and this results in a substantial loss of the hate speech class features because of the imbalanced dataset.

To fix this problem, in future work we should either try to "cherry-pick" ngrams that show in both the hate speech class and offensive language class or use a different set of features. When comparing the micro-average F1 score for all classes Davidson et al. (2017) has an F1 score of 0.9 while our best model is the logistic regression with the score of 0.85. Our SVM model has an F1 score of 0.8, which is a considerable drop compared to Davidson et al. (2017), while our active learning models F1 score is only 0.03 behind SVM. Considering that the Davidson et al. (2017) was trained using a substantially larger set of features, it is expected to have better results. That is why we are going to compare our active learning model to our own implementations of SVM and logistic regression that use the same set of features. All our testing was done on an Intel(R) Core(TM) i5-7200U CPU with 16 GB of RAM.

Table 3. shows the difference in F1 scores and time needed to train each model. As we can see the logistic regression class outperforms SVM and active learning model by a large margin. Even though the F1 scores for hate speech class are the same for all three models, the biggest difference is in detecting clean text with F1 scores varying from

Table 3: Table shows average F1 scores for every class individually and micro averaged F1 for all classes together for each of our models. It also contains estimated average time needed to train one of ten K folds in our cross-validation.

	F1(hate)	F1(offensive)	F1(clean text)	F1(all)	Training time(minutes)
Logistic Regression	**0.26**	**0.92**	**0.70**	**0.85**	**6 min**
SVM	**0.26**	0.90	0.55	0.80	70 min
Active Learning	**0.26**	0.87	0.46	0.77	16 min

0.46 to 0.7. Even though the clean text class has the biggest variance between the models, the biggest impact on the total F1 score comes from the offensive language class. The reason for this is the imbalanced dataset where over 70% of the data is classified as offensive language. The result of the imbalanced dataset is that offensive language class has a considerably higher recall than precision while the other two classes have a higher precision than recall. Even though our active learning model has worse results than the other two models, it is important to mention that it has the smallest differences in precision and recall for each class. Why is this the case? As we mentioned, we chose active learning model as a way to train a model on a much smaller dataset. The model was trained on around 8000 and tested on approximately 2500 tweets. Because of that, the difference in the dataset distribution was not as impactful as it was on the other two models that were trained on 20000 tweets. We believe that given a balanced dataset, all our models would perform much better, but active learning would not need as much data to train itself, which is a significant step forward.

Table 4: Table contains F1 scores for four different models. Column F1(hate) shows F1 scores calculated only on hate speech class while column F1(all) scores are calculated as micro average F1 scores on all three classes.

	F1(hate)	F1(all)
Davidson et al.	**0.51**	**0.90**
Logistic Regression	0.26	0.85
SVM	0.26	0.80
Active Learning	0.26	0.77

6. Conclusion

In the modern-day society, hate speech is becoming a great problem, but detecting it could be considered an even greater problem. It is hard for a human to detect the difference between hate speech and offensive language, and teaching a machine to solve that problem is an interesting and an ambitious project.

We know that detection methods based on lexical features lack precision when it comes to separating hate speech from offensive language or detecting hate speech when no offensive language is used. Some machine learning methods give better results and that was our point of interest that we used when we started working on this paper.

We used the dataset created by Davidson et al. (2017) that contains a great imbalance in the distribution of the classes. To solve this problem, we implemented a semi-supervised model based on active learning. We wanted to get the same or better results than other baseline models while training on a much smaller dataset. Our implementation of an active learning model shows promising results while being simple and using only a small set of features. In future, we would like to further explore the use of active learning with a different set of features, and creating a much more computationally expensive active learning algorithm that takes into account the whole dataset and not just the training set.

References

Yi-Wei Chen and Chih-Jen Lin, 2006. *Combining SVMs with Various Feature Selection Strategies*, pages 315–324. Springer Berlin Heidelberg.

Thomas J Davidson, Dana Warmsley, Michael W. Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *ICWSM*.

Tong Luo, Kurt Kramer, Dmitry B. Goldgof, Lawrence O. Hall, Scott Samson, Andrew Remsen, and Thomas Hopkins. 2005. Active learning to recognize multiple types of plankton. *Journal of Machine Learning Research*, pages 589–613, April.

2009. Natural language toolkit. `https://www.nltk.org/`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830.

2009. Porter stemmer. `https://www.nltk.org/_modules/nltk/stem/porter.html`.

Zeerak Waseem, Thomas Davidson, Dana Warmsley, and Ingmar Weber. 2017. Understanding abuse: A typology of abusive language detection subtasks. In *Proceedings of the First Workshop on Abusive Language Online*, pages 78–84. Association for Computational Linguistics.

Bishan Yang, Jian-Tao Sun, Tengjiao Wang, and Zheng Chen. 2009. Effective multi-label active learning for text classification. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 917–926. ACM.

A Self-Attentive Similarity-Based Approach for Community QA Ranking

Antonio Šajatović, Tome Radman, Lukrecija Puljić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{antonio.sajatovic,tome.radman,lukrecija.puljic}@fer.hr

Abstract

One of the challenges in maintaining a community question answering service is ranking the answers and related questions by their relevance to the original question. We propose a neural network model which exploits sentence similarity in determining the ranks of given answers or related questions. The model consists of a sentence encoder and a self-attention mechanism applied on interaction features between the calculated encodings. Two different encoders are presented: a hierarchical convolutional neural network and a bidirectional gated recurrent unit (Bi-GRU) network with mean and max pooling. Training and evaluation were done on the dataset for Task 3 of SemEval-2017. Both approaches are shown to give promising results through experimental analysis.

1. Introduction

Online forums are by far one of the most practical ways for people to ask questions and get immediate answers from their peers. The amount of data that is accumulating on such online sites is constantly growing and already presents a valuable data source for various purposes. The main problem with this source of data is that it is highly unstructured, which means it is not straightforward to find out whether this particular question has already been asked or if there is a similar question already and to extract relevant information from available answers.

In this paper, we describe a system to automatically find relevant content from an online forum. The system is designed to solve two tasks as proposed in the SemEval 2017 competition. The first task, called subtask A, is *Question-Comment Similarity*. The system is given a question and the first ten answers in its thread. The goal is to rank these answers according to their relevance to the given question. For the subtask B, the system is given a question and the first ten related questions from the forum, as retrieved by a search engine. The goal is to rank these questions according to their relevance to the given question. This is *Question-Question Similarity* task.

We experimented with GRU (Cho et al., 2014) and CNN encoders in the modified conventional architecture for training NLI data (Bowman et al., 2015). Learning to rank was approximated via a pointwise approach, i.e., learning a binary classifier that can tell if an answer is relevant or non-relevant for a question in subtask A, and if two questions are similar or not in subtask B.

The rest of the paper is organized as follows: section 2 describes relevant work, section 3 explains systems architecture, section 4 describes the datasets and section 5 presents the experiments and their results.

2. Related Work

The motivation behind SemEval 2017 Task3 is to automate the process of finding good answers to new questions in a Community Question Answering (CQA) discussion forum. To tackle this problem, the researchers proposed the two previously mentioned tasks.

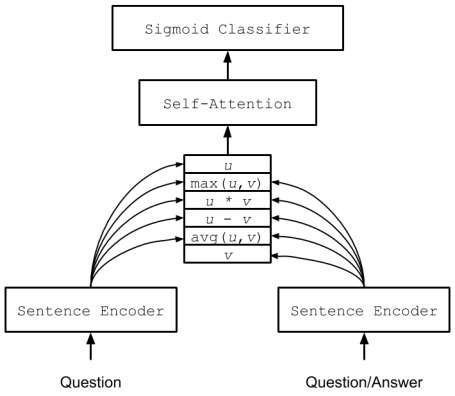


Figure 1: Overall system architecture.

In the past, *Question-Question Similarity* was tackled using various approaches like using question topic and focus (Duan et al., 2008) or using LDA topic language model that matches the questions both at the term and topic level (Zhang et al., 2014). Syntactic structure features were widely used for some of the top systems that participated in SemEval-2016 Task 3 (Nakov et al., 2017).

The most common approach for *Question-Answer Similarity* was to match the syntactic structures of the question and the candidate answer, particularly in the top systems that participated in SemEval-2016 Task 3 (Nakov et al., 2017). A similar approach was used in (Filice et al., 2017), one of the top systems on both subtasks.

Our approach differs from the previous ones in several ways: we use the same neural network model for both subtasks and we hypothesize that slightly modified architecture for NLI and neural network classifiers used in (Conneau et al., 2017) are beneficial for our task, especially given the pointwise approach. As far as we are aware, our model is the first one that uses a self-attention mechanism as described in (Shen et al., 2017).

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 96–100, University of Zagreb,
Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

3. Architecture

The overall system architecture is shown in Figure 1. The inputs are either question/answer embeddings (subtask A) or question/question embeddings (subtask B). Shared weight encoders output a representation for the question u and the answer v. Then, six matching methods are applied to extract the relations between u and v :

- question encoding u,
- element-wise maximum $max(u, v)$,
- element-wise average $avg(u, v)$,
- element-wise product $u * v$,
- element-wise difference $u - v$
- question/answer encoding v

These six vectors representations are then concatenated and reshaped, such that their last dimension matches the self-attention layer size, and fed into the self-attention layer. The resulting vector is fed into a binary logistic regression classifier. As mentioned in the previous section, we hypothesize that multiple matching methods are better able to capture the relevant information for similarity between the inputs and we hope that the model can learn the importance of each via the self-attention mechanism.

The sentence encoders we employ are Bidirectional GRU and Hierarchical ConvNet with max or mean pooling, as described in (Conneau et al., 2017). All three achieve near state-of-the-art performance on the STS14 - Semantic Textual Similarity (Agirre et al., 2014) and SICK-R (Marelli et al., 2014) datasets, which are closely related to both subtasks A and B. Detailed model descriptions that we followed in our implementations are available in (Conneau et al., 2017). In the following subsections, each model is shortly described.

3.1. Hierarchical ConvNet

Hierarchical ConvNet is a streamlined version of the AdaSent (Zhao et al., 2015) convolutional architecture consisting of only four convolutional layers. The model captures hierarchical abstractions of an input sentence in a fixed-size representation by computing a mean-pooling or max-pooling operation over the hierarchical feature maps. The final representation $u = [u_1, u_2, u_3, u_4]$ concatenates representations at different levels of the input sentence, as shown in Figure 2.

3.2. BiGRU with Mean and Max Pooling

A common approach in encoding sequences is using recurrent neural networks. We opted for using a bidirectional RNN with gated recurrent unit (GRU) cells (Cho et al., 2014). The model outline can be seen in Figure 3. For each word in the output sequence, a bidirectional GRU outputs a vector h_t, which is a concatenation of backward and forward GRU hidden states:

$$\overrightarrow{h_t} = \overrightarrow{GRU}_t(w_1, ..., w_T) \quad (1)$$

$$\overleftarrow{h_t} = \overleftarrow{GRU}_t(w_1, ..., w_T) \quad (2)$$

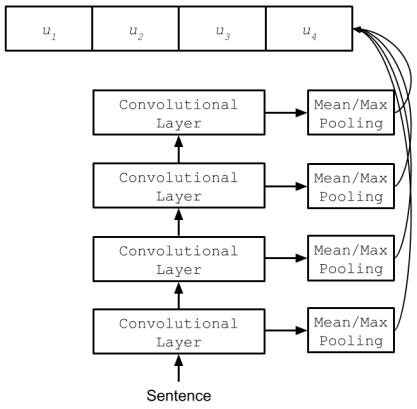


Figure 2: Hierarchical ConvNet encoder.

$$h_t = \left[\overrightarrow{h_j}, \overleftarrow{h_j}\right] \quad (3)$$

Instead of only using the output from the last timestep, vectors $(h1, ..., h_T)$ are aggregated using mean or max pooling: each dimension of the final vector contains the average or maximum value of that dimension from the Bi-GRU outputs at each timestep. This enables the model to utilize information from different timesteps more successfully.

Our final encoder uses two Bi-GRU networks, with outputs of the first network being fed as inputs into the second network. The outputs from both networks are passed through a pooling layer. The final encoding is a concatenation of the pooled outputs from both networks (Figure 4).

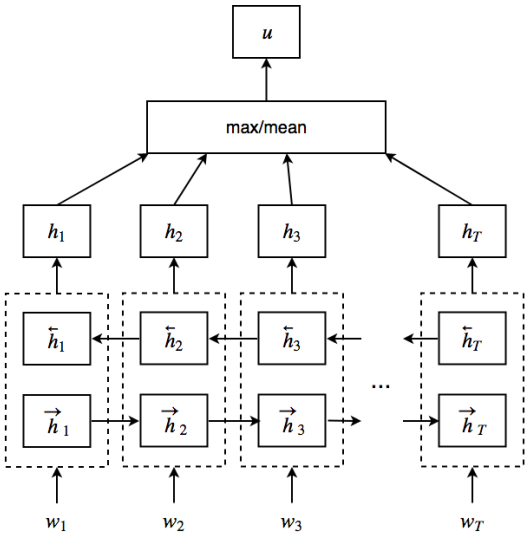


Figure 3: A detailed overview of the Bi-GRU network with mean/max pooling used as a part of the encoder.

3.3. Self-Attention

Self-attention, also called intra-attention, has been used successfully in a variety of tasks, with the most recent success in neural machine translation (Vaswani et al.,

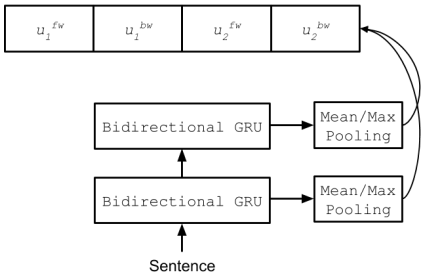


Figure 4: Bidirectional GRU encoder.

2017). Inspired by such success, we have implemented the *"source2token"* multidimensional self-attention mechanism as defined in (Shen et al., 2017). Source2token self-attention explores the dependency between x_i and the entire sequence of vectors $\boldsymbol{x}$, compressing $\boldsymbol{x}$ (in our case, the sub-representation vectors) into a single vector. For each x_i, we calculate a feature-wise score vector:

$$z_i = f(x_i) = W^{(2)} ELU(W^{(1)} x_i). \quad (4)$$

Attention weights are calculated using the following equation:

$$\alpha_i = p(z = i|\boldsymbol{x}) = softmax(z_i). \quad (5)$$

The output of source2token self-attention for $\boldsymbol{x}$ is the context vector c:

$$c = \sum_{i=1}^{n} \alpha_i \odot x_i. \quad (6)$$

The resulting context vector is finally fed into a single sigmoid classifier.

4. Dataset

We used the official data from SemEval 2017 competition, which is publicly available. The data for both subtasks were gathered from the Qatar Living forum. In the subtask A, for each of the forum topics, top 10 answers were taken and manually annotated as being "Good", "PotentiallyUseful" or "Bad" regarding the thread question being asked. For the subtask B, the dataset consisted of questions and the related thread questions which were annotated as being either a "PerfectMatch", "Relevant" or "Irrelevant" with respect to the original one. From subtask A's labels, "Bad" and "PotentiallyUseful" were both taken as being bad during evaluation, while for subtask B, both "PerfectMatch" and "Relevant" were considered as good. Statistics about the dataset are shown in Table 1.

Category	Train	Dev	Test
Relevant Answers	1900	2440	3270
Good	6651	818	1329
Potentially Useful	3110	413	456
Bad	8139	1209	1485
Related Question	2669	500	700
Perfect Match	235	59	81
Relevant	848	155	152
Irrelevant	1586	286	467

Table 1: Dataset statistics.

5. Experiments

In this section, we first describe the training setup in detail and then discuss the obtained results.

5.1. Training Setup

We initialize the word embeddings by GloVe 6B pre-trained vectors (Pennington et al., 2014). The Out-of-Vocabulary words in the training set are initialized to an all-zero vector. The models were trained for 30 epochs as we observed that training any longer would lead to overfitting, even when using Dropout (Srivastava et al., 2014). On subtask A we used the Adadelta optimizer (Zeiler, 2012), while on subtask B we used the Adam optimizer (Kingma and Ba, 2014), due to convergence, both with default settings. Batch size was set to 32 in both subtasks. In order to find the optimal hyperparameter values, we tested several values from an arbitrarily chosen range for each hyperparameter and chose those that minimized the network loss on the subtask A development dataset. The optimal values are shown in Table 2. Grid search method was not feasible in our case due to a large parameter search space. All three models are implemented using Keras (Chollet and others, 2015) and if not specified, all other hyperparameters were set to default Keras values. The models were run on the Google Colaboratory platform[1] with GPU runtime enabled.

Hyperparameter	Value
Dropout rate	0.25
BiGRU hidden size	300
Sentence timestep	100
CNN filter size	300
CNN kernel size	3
FC layer size	300
Activation function	ELU
Word embedding size	300
Pooling method	max
Loss function	binary cross-entropy

Table 2: Model hyperparameters.

5.2. Results

Tables 3 and 4 show the performances of all presented models for subtask A and B, respectively, obtained using the official evaluation script. The organizers chose *mean average precision* (MAP) as the official metric for this task. MAP is calculated on a list of ranked answers or related questions, sorted by classification probabilities in descending order.

In subtask A, all of our systems managed to outperform the baseline by a wide margin, with *bigru_maxpool* being the most successful, achieving a MAP score of 82.60. By conducting a permutation test, we were able to show that the difference between *bigru_maxpool* and the convolutional models is statistically significant with a 0.05 level of significance, as shown in Table 5.

[1] https://colab.research.google.com

Our models failed to outperform the IR baseline for subtask B. This could be attributed to our decision to do model selection only on subtask A and reuse the hyperparameters on subtask B, in an attempt to create a more general model applicable to various sentence similarity tasks. The best-performing model was *cnn_maxpool* (MAP 33.68). Statistical significance results are shown in Table 6.

System	MAP	Acc	F1
bigru_avgpool	80.84	67.58	62.00
bigru_maxpool	**82.60**	65.43	54.80
cnn_avgpool	79.82	65.29	56.70
cnn_maxpool	80.18	65.56	58.22
KeLP	**88.43**	73.89	69.87
Beihang-MSRA	88.24	51.98	68.40
SwissApls	86.24	61.30	43.30
Baseline 1 (IR)	72.61	-	-
Baseline 2 (random)	62.30	52.70	62.54
Baseline 5 (TF-IDF)	0.43	0.63	0.49

Table 3: Subtask A results.

System	MAP	Acc	F1
bigru_avgpool	29.22	63.52	31.26
bigru_maxpool	33.36	60.00	33.33
cnn_avgpool	30.68	61.14	30.20
cnn_maxpool	**33.68**	52.27	34.98
SimBow	**47.22**	52.39	42.37
LearningToQuestion	46.93	18.52	31.26
KeLP	46.66	69.20	50.64
Baseline 1 (IR)	41.85	-	-
Baseline 2 (random)	29.81	34.77	30.00
Baseline 5 (TF-IDF)	0.43	0.40	0.57

Table 4: Subtask B results.

model	bigru_avgpool	bigru_maxpool	cnn_avgpool	cnn_maxpool	Baseline IR	Baseline TF-IDF
bigru_avgpool		=	=	=	*	*
bigru_maxpool			*	*	*	*
cnn_avgpool				=	*	*
cnn_maxpool					*	*

* Statistically significant, $p \leq 0.05$
= Not significant, $p > 0.05$.

Table 5: Significance of MAP differences between system pairs for subtask A.

6. Conclusion

We presented a deep neural network architecture for relevancy ranking in community question answering services, using two kinds of sentence encoders: a hierarchical convolutional neural network and a bidirectional GRU with mean/max pooling. A self-attention mechanism was used to assign importance to different interaction features. The models were evaluated on SemEval-2017 task 3 dataset, achieving a MAP score of 82.60 on subtask A and 33.68 on subtask B.

model	bigru_avgpool	bigru_maxpool	cnn_avgpool	cnn_maxpool	Baseline IR	Baseline TF-IDF
bigru_avgpool		=	=	=	*	*
bigru_maxpool			=	=	*	*
cnn_avgpool				=	*	*
cnn_maxpool					*	*

* Statistically significant, $p \leq 0.05$
= Not significant, $p > 0.05$.

Table 6: Significance of MAP differences between system pairs for subtask B.

Our goal was to create a general system reusable in different sentence similarity scenarios. Hence, we only used the dataset for subtask A for model selection, which resulted in a drop in performance on subtask B.

For future improvements, more focus should be given to finding different ways to combine the encoded sentences and it would also be worth trying a character-based model.

Acknowledgements

We thank dr.sc. Mladen Karan[2] from TakeLab for providing us with the adequate statistical significance tests.

References

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2014. Semeval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 81–91.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.

François Chollet et al. 2015. Keras. `https://keras.io`.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Huizhong Duan, Yunbo Cao, Chin-Yew Lin, and Yong Yu. 2008. Searching questions by identifying question topic

[2] `http://takelab.fer.hr/mladen/`

and question focus. *Proceedings of ACL-08: HLT*, pages 156–164.

Simone Filice, Giovanni Da San Martino, and Alessandro Moschitti. 2017. Kelp at semeval-2017 task 3: Learning pairwise patterns in community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 326–333.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, Roberto Zamparelli, et al. 2014. A sick cure for the evaluation of compositional distributional semantic models. In *LREC*, pages 216–223.

Preslav Nakov, Doris Hoogeveen, Lluís Màrquez, Alessandro Moschitti, Hamdy Mubarak, Timothy Baldwin, and Karin Verspoor. 2017. Semeval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 27–48.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2017. Disan: Directional self-attention network for rnn/cnn-free language understanding. *arXiv preprint arXiv:1709.04696*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Kai Zhang, Wei Wu, Haocheng Wu, Zhoujun Li, and Ming Zhou. 2014. Question retrieval with high quality answers in community question answering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 371–380. ACM.

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence model. In *IJCAI*, pages 4069–4076.

Who is More Sentimental: Comparing SVR and LSTM on Predicting Tweet Sentiment

Dunja Vesinger, Nikola Vrbanić, Vedran Ivanušić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{dunja.vesinger,nikola.vrbanic,vedran.ivanusic}@fer.hr

Abstract

Sentiment analysis is a task of detecting the polarity or attitude which is expressed through language and it's nowadays widely used in various areas. Most recent work in this field focuses on deep learning models while the use of more traditional approaches such as support vector regression is decreasing. The traditional models are mostly used as baselines and trained with the same features as deep learning models. However, these features are not optimal for training such models. This work aims to present a fair comparison between a deep learning model and a traditional approach. This is done by designing a separate set of features for each model using the same resources. Finally, we show that we can further improve the performance by combining the two approaches and achieve performance comparable to state-of-the-art models.

1. Introduction

Sentiment is a subjective opinion or attitude a person holds towards something. Detecting sentiment has become a vital task in many areas, especially those concerning marketing or customer service, and it has also been found useful in clinical medicine. However, sentiment can be expressed in a wide variety of ways which makes sentiment analysis a challenging task.

This paper studies the task of sentiment analysis in the realm of Twitter - a microblogging platform which enables users to post short sections of text called tweets. The task is framed as a regression problem where the aim is to represent each tweet with a real value score. Scores range between 0 (most negative) and 1 (most positive). Examples of tweets are given in Table 1.

This problem was featured as one of the tasks in the SemEval competition (Mohammad et al., 2018) titled Valence Regression. The competition resulted in 37 systems featuring several machine learning models. Majority of them used Long Short-Term Memory Networks (LSTMs) or Bidirectional Long Short-Term Memory Networks (Bi-LSTMs). The second most used model was Support Vector Regression (SVR) or Support Vector Machine (SVM). We observed that competition papers published so far mostly used SVR in a combination with deep learning models rather than as a standalone model. Furthermore, the baseline provided by the organizers is a rather uncompetitive SVM trained solely using word unigrams which was easily surpassed by the contestants.

It is evident that previous results speak in favor of deep learning models which significantly outperform the SVM baseline. However, the proposed baseline only uses word unigram features which are not adjusted to the task of sentiment analysis and hardly provide an optimal set of features for an SVM model.

The goal of our research is to provide a fairer comparison of deep learning models to models based on support vectors in this field. Furthermore, we compare the performance of the combined model which uses deep learning embeddings and SVR with handcrafted features to the performance of the individual models.

2. Related Work

SemEval competition introduced several approaches to the task of sentiment analysis in tweets. There were 37 entries for the task of Valence Regression. Unfortunately, only a handful of them have been published to this date. It is important to emphasize that none of the published models were specifically designed for Valence Regression. They were all used in several, if not all, tasks related to detecting affect in tweets featured at SemEval 2018.

Park et al. (2018) used a combination of three different embeddings produced by deep learning models trained large distant supervision corpora. Additionally, they included a handful of handcrafted features such as the number of uppercase words and the sum of emoticon valence scores, but decided to refrain from using sentiment lexicons. We did not follow this approach as sentiment lexicons proved to be useful in other works, but we tested several of their handcrafted features as well as similar deep learning models based on Bi-LSTMs and GloVe representations (Pennington et al., 2014).

Other approaches such as Mohammed and Moreno (2018) and Baziotis et al. (2018) also opted for deep learning methods, but did not yield as good results as Park et al. (2018).

Most of the work prior to the 2018 SemEval competition framed sentiment analysis as a classification task. Since the goal is very similar, it is reasonable to expect that features used for classification would perform well on the regression task as well.

Kouloumpis et al. (2011) studied the performance of several sets of handcrafted features on Twitter data. They used bag-of-words embeddings based on n-grams, features based on sentiment lexicons, part-of-speech (POS) tags and Twitter specific features such as emoticons, abbreviations and uppercase words. POS tags seemed to deteriorate the performance of their model so we opted not to explore these

Text Analysis and Retrieval 2018: Course Project Reports (TAR 2018), pages 101–105, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2018.

Table 1: Examples of tweets and their sentiment scores.

Tweet	Sentiment Score
It's meant to be!! #happy #happy	0.966
"When I get sad, I stop being sad and be awesome instead." - Barney Stinson	0.532
I can't pull myself out of depression	0.054

features in our work, but we included several features based on semantic lexicons and structures specific for Twitter.

A more detailed ablation study was performed by Mohammad et al. (2013a). It highlighted features based on sentiment lexicons as the most informative ones. They also observed the appearances of negations such as "not" or "don't" which positively affected the model performance. We further explore both suggested feature sets in our work.

We combined several of the aforementioned approaches to design comparable feature sets for both SVR and Bi-LSTM. Furthermore, we explored different methods of combining these two models and observed the performance of the joint model.

3. System Description

We present a comparison of three different models SVR, Bi-LSTM and a combination of these two models. Our SVR uses several handcrafted features as combined with word embedding features. Bi-LSTM also uses word embeddings with some additional features retrieved from sentiment lexicons. We also tested the combination of these two models, where we used Bi-LSTM for creating the embeddings used for training the SVR model. A detailed overview of the models is given in the sections which follow.

All models were trained solely on the dataset provided by the SemEval 2018 competition (Mohammad et al., 2018). For feature engineering, we used the proposed training set for training the models and evaluated the performance of the given features using the development set.

3.1. Preprocessing Tweets

We performed the same preprocessing on tweets use for all proposed models. Tweets were tokenized and POS-tagging was performed using the tools specialized for Twitter data proposed by Gimpel et al. (2011) and Owoputi et al. (2012). We used the POS tags to remove usernames and to separate hashtags and emoticons from the rest of the text in tweets for further feature extraction. Since the acquired tokens and POS-tags were not ideal (e.g. hashtags are sometimes tagged as emoticons), we performed some additional processing to correct the imperfections.

Despite previous research on similar tasks which suggests that detecting uppercase words is an informative feature, we found that it deteriorated the performance of the system for this particular task so we transferred all words to lowercase.

For the Bi-LSTM model, each tweet was reshaped to fit the average length of a tweet, 20 words, either by cutting it or padding it with a filler vector.

3.2. Sentiment Lexicons

Several features based on word sentiment were used in both of the systems. Exact features used are discussed in the following sections, but all of them were extracted from the same sentiment lexicons. We used four different lexicons: VADER (Hutto and Gilbert, 2014), AFINN (Årup Nielsen, 2011) and NRC lexicons for words (Mohammad, 2018) and hashtags (Kiritchenko et al., 2014; Mohammad et al., 2013b; Zhu et al., 2013). Sentiment scores of all lexicons were scaled to range between -1 (most negative) to +1 (most positive) before usage.

Tweets nowadays commonly contain emojis - small images of emoticons which can be inserted in a text and are encoded as special Unicode characters. Unfortunately, none of the above sentiment lexicons includes scores for emojis. We evaluated two different approaches to handling emoji sentiment: we used a separate dictionary created for emoji sentiment (Kralj Novak et al., 2015) and we tried manually mapping the emojis to their nearest textual emoticon representations and using the sentiment scores for these representations from sentiment lexicons mentioned above. Surprisingly, the mapping to textual representations resulted in better performance so we used this as a primary solution and only included emoji sentiment from (Kralj Novak et al., 2015) if no mapping was available.

3.3. Word Embeddings

We used GloVe word embeddings pre-trained on Twitter data (Pennington et al., 2014). SVR features included GloVe word representations of length 25 which were compressed into a single vector using sum polling. Max polling and min pooling both dropped the performance of the SVR compared to sum pooling. Longer GloVe representations also had a negative effect on the predictions of the SVR.

On the other hand, the Bi-LSTM based model used GloVe's 50 dimension representations as inputs.

In a composed model word embeddings acquired via sum pooling were discarded from the SVR features and replaced by the embeddings the Bi-LSTM produced.

3.4. Features Used for Training SVR

The final SVR model uses 36 features. The word embedding features described in Section 3.3. form a set of 25 features and the remaining 11 are compromised of various handcrafted features. The complete list of features is presented in Table 2.

All features were scaled by removing the mean value and scaling to unit variance before being passed to the SVR.

Table 2: A list of features used by the SVR.

SVR Features
Word embedding features
Sum of word sentiment scores
Sum of hashtag sentiment scores
Sum of emoticon and emoji sentiment scores
Sum of all sentiment scores
Sum of all negative sentiment scores
Highest sentiment score
Lowest sentiment score
Sentiment score of the last negative word
Number of negations used
Presence of question marks
Presence of exclamation marks

Centering and scaling were performed independently on each feature and its parameters were always fitted exclusively on the dataset used for training the model. We also tried averaging features which were calculated by aggregating sentiments scores, but it deteriorated the performance so we dismissed it.

A sentiment score for each token was calculated as the average value of sentiments from four sentiment lexicons listed in Section 3.2.. Since not all lexicons contain all kinds of tokens (e.g. hashtags, emoticons, slang), only lexicons which contained the given token were taken into account. If none of the lexicons contained the token, sentiment score of the token was set to zero.

Apart from sentiment lexicons, we also used a predefined list of commonly used negations in the English language such as "not" or "don't". The list is provided by the VADER lexicon (Hutto and Gilbert, 2014). Negations are usually marked as neutral by the sentiment lexicons, but may reverse the sentiment of the other words in the tweet.

It is interesting to notice several of the listed features revolve around words which have negative sentiment. We also tried including their positive counterparts, but this lowered the performance of our model. Furthermore, although previous work uses features containing the number of exclamation marks or question marks, we found that indicator variables which merely capture the presence of these punctuation marks result in better performance.

3.5. Bi-LSTM Features and Architecture

For our deep learning model, we opted for an LSTM based model. We tried out multiple models consisting of 1 to 3 LSTM layers, each consisting of 16,32,64 or 128 neurons and using either a normal LSTM layer or bidirectional LSTM layer and given multiple combinations of features. The best performing Bi-LSTM model is a single layer bidirectional LSTM layer with 32 neurons followed by a fully connected sigmoid neuron that uses 55 features. GloVe's 50-dimensional word embedding as described in Section 3.3. forms the first 50 features. The remaining 5 features are the sentiment scores of each individual sentiment lexicon we used, as well as their average value. The reason behind the best model's architecture being so simple is the size of the dataset. We discuss this in more detail in Section 4.2.

3.6. Combined Model

The combined model uses the same handcrafted features as the SVR, while the word embedding features were replaced by the output of the last Bi-LSTM layer. The reasoning behind this is that we assume that the Bi-LSTM layer learned representations which can be useful in calculating the sentiment score. Since the Bi-LSTM was trained to predict the sentiment scores of each tweet, it is intuitive that the embeddings it produced will contain more discriminative information than the ones acquired through sum pooling of GloVe representations.

4. Experimental Setup

We used the dataset provided by the SemEval 2018 competition (Mohammad et al., 2018) for subtasks *Valence Regression of the Affect in Tweets* task. The English language portion of this dataset was used. It is divided into three subsets: the training set contains 1181 tweets while development and test set have 449 tweets each. In order to be able to compare our models to the results featured in previous work, we used the same dataset split.

For the purposes of feature engineering and hyperparameter adjustment, the training set was used to train the model and the performance was evaluated on the development subset. The final performance scores featured in the results section were acquired by training the models on joint train and development subset while the evaluation was done on the test subset.

The performance of all models was evaluated by calculating the Pearson correlation coefficient between the correct and the predicted output on the test set. This measure was chosen as it was the official measure of the SemEval competition and enables us to effectively compare our results to previous work.

4.1. SVR Training

Feature engineering for the SVR was performed using grid search with 3-fold cross-validation. The hyperparameters for each feature set were determined separately using grid search. The model was then trained on the whole training set and evaluated on the development set.

The hyperparameters of the final models featured in Table 3 and Table 4 were determined by grid search using 3-fold cross-validation on the combined training and development dataset. The final model was then trained on the entire train and development set and evaluated on the test set.

4.2. LSTM Training

Since the dataset given for this task is pretty small compared to standard deep learning datasets the model was very susceptible to overfitting. In order to combat this, and pull the most out of the data we had available we used a dropout rate of 50% on the BiLSTM layer and early stopping on the development set. After optimizing all hyperparameters and evaluating the point on which the model first starts

Table 3: Comparison of several SVR models trained on different sets of features.

Model	Pearson
SVR with handcrafted features	0.769
SVR with Bi-LSTM embeddings	0.771
SVR with handcrafted features and simple word embedddings	0.781
SVR with handcrafted features and Bi-LSTM embeddings	**0.800**

Table 4: Comparison between SVR and Bi-LSTM.

Model	Pearson
SVR	0.781
Bi-LSTM	0.778
Combined model	**0.800**

overfitting, we combined the train and development set and trained the model to that same point with all data available.

5. Results

We compare two models: the SVR trained using a combination of handcrafted features and simple word embeddings and a Bi-LSTM trained on word embeddings and several sentiment features. The results are shown in Table 4. Both models use pre-trained GloVe embeddings specifically created for Twitter data. Furthermore, both of them use the same sentiment lexicons for additional features. However, we did not insist on using the identical set of features to train both models as we considered this to be unfair. We aimed to compare both models at their best, so instead of using the same features for both models, we used the same resources to come up with an optimal set of features which suits each model.

Since successful previous works commonly combined several models to produce the best results, we also tested the combination of SVR and Bi-LSTM. We removed the simple word embeddings used for training the SVR and replaced them with the embeddings produced by the last layer of our Bi-LSTM model. The performance of this model is also shown in Table 4. While the SVR and the Bi-LSTM produced comparable results, combining them produced a relatively minor improvement in the performance. This situation is commonly seen in previous work as well. However, we also performed permutation testing which showed that the differences are not statistically significant at α = 0.05.

As we were interested to find out what is the cause of such minor improvement, considering both models showed good performance on their own, we also conducted a series of experiments which aim to compare several models trained on subsets of the features used in our final model.

We trained four SVR models. The first one only used the handcrafted features and no word embeddings. The second one used solely the word embeddings produced by the Bi-LSTM. The third one is the SVR with handcrafted features and simple word embeddings while the final model uses handcrafted features and embeddings produced by the Bi-LSTM. Their scores are shown in Table 3. It is clear that all the feature sets produced models of comparable performance. However, combining them brings no major improvement. This suggests handcrafted features and word embeddings are highly dependent, but not completely interchangeable.

Table 5: Comparison between several models used in SemEval-2018 competition.

Model	Pearson
Duppada et al. (2018)	**0.873**
Park et al. (2018)	0.860
Baziotis et al. (2018)	0.851
Mohammed and Moreno (2018)	0.828
Our Work	0.800
SVM Unigrams Baseline	0.585
Random Baseline	0.031

Finally, we compared our most successful model to other models from the competition. We included two baselines provided by the organizers, the best performing system and the models mentioned in Section 2.

Unfortunately, the model behind the best performing system has not yet been published. The best performing models published up to date are Park et al. (2018) and Baziotis et al. (2018). However, these models are not completely comparable to ours as both of them use additional training data acquired through distant supervision or task transfer. The work of Mohammed and Moreno (2018) and the baseline models were trained on the same data as our model and therefore present a fairer comparison.

6. Conclusion

In this paper we compare SVR and Bi-LSTM on the task of predicting the sentiment score of tweets. The models achieved comparable performances on the SemEval2018 dataset. Furthermore, we tested a combined model which uses handcrafted features and word embeddings produced by the Bi-LSTM. This model shows a small improvement in performance compared to the individual models. This suggests that handcrafted features and word embeddings we used are highly correlated, but not completely dependent. In future work we would like to further explore the correlation between word embedding and handcrafted features.

References

Christos Baziotis, Nikos Athanasiou Alexandra Chronopoulou, Athanasia Kolovou, Georgios Paraskevopoulos, Nikolaos Ellinas, Shrikanth Narayanan, and Alexandros Potamianos. 2018. NTUA-SLP at semeval-2018 task 1: Predicting affective content in tweets with deep attentive rnns and transfer learning.

Venkatesh Duppada, Royal Jain, and Sushant Hiray. 2018. Seernet at semeval-2018 task 1: Domain adaptation for affect in tweets. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, LA, USA.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

C.J. Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media*.

Svetlana Kiritchenko, Xiaodan Zhu, and Saif M. Mohammad. 2014. Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research (JAIR)*, 50:723–762.

Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. 2011. Twitter sentiment analysis: The good the bad and the omg! In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*.

Petra Kralj Novak, Jasmina Smailović, Borut Sluban, and Igor Mozetič. 2015. Sentiment of emojis. *PLoS ONE*, 10(12).

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013a. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets.

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013b. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. In *Proceedings of the seventh international workshop on Semantic Evaluation Exercises (SemEval-2013)*, Atlanta, Georgia, USA, June.

Saif M. Mohammad, Felipe Bravo-Marquez, Mohammad Salameh, and Svetlana Kiritchenko. 2018. Semeval-2018 Task 1: Affect in tweets. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, LA, USA.

Saif M. Mohammad. 2018. Obtaining reliable human ratings of valence, arousal, and dominance for 20,000 english words. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Jabreel Mohammed and Antonio Moreno. 2018. Eitaka at semeval-2018 task 1: An ensemble of n-channels convnet and xgboost regressors for emotion analysis of tweets.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, and Nathan Schneider. 2012. Part-of-speech tagging for twitter: Word clusters and other advances. *Technical Report*.

Ji Ho Park, Peng Xu, and Pascale Fung. 2018. Plusemo2vec at semeval-2018 task 1: Exploiting emotion knowledge from emoji and #hashtags.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Xiaodan Zhu, Svetlana Kiritchenko, and Saif M. Mohammad. 2013. Nrc-canada-2014: Recent improvements in sentiment analysis of tweets. In *Proceedings of the seventh international workshop on Semantic Evaluation Exercises (SemEval-2013)*, Atlanta, Georgia, USA, June.

Finn Årup Nielsen. 2011. A new anew: Evaluation of a word list for sentiment analysis in microblogs. In *Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts'*.

Author Index