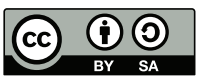
Text Analysis and Retrieval 2019

Course Project Reports

University of Zagreb

Faculty of Electrical Engineering and Computing



Publisher:

University of Zagreb, Faculty of Electrical Engineering and Computing

Organizers & editors:

Mladen Karan
Jan Šnajder

Reviewers:

Karlo Brajdić
Mladen Karan
Tome Radman
Antonio Šajatović
Jan Šnajder
Dunja Vesinger

ISBN 978-953-184-261-7

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 sixth 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, with a focus on practical applications, all while being in line with the best practices in the area.

These outcomes are 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 16 projects, which are the work of 48 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 fact checking, partisanship detection, humor analysis, knowledge extraction from medical texts, depression detection, song lyrics classification, and stock market prediction, to name a few. With respect to methods, the trend of increasing popularity for deep learning is present for several years and continues this year, with most teams using some sort of deep learning. We believe part of the reason for this is the increasing availability of deep learning libraries that are simple to use and the recent development of contextualized word embedding models that provide a high quality pretrained neural word representations useful for almost any task. Some of the teams really went above and beyond by implementing custom neural architectures with custom loss functions etc.

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. Similarly to previous editions of the course, our intuitions about this approach were 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 and Jan Šnajder

Contents

How Soon Can We Detect Depression?

Luka Banović, Valentina Fatorić, Daniel Rakovac

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{luka.banovic,valentina.fatoric,daniel.rakovac}@fer.hr

Abstract

This paper presents an approach for finding an optimal amount of data per subject to detect depression as early as possible. In addition, we propose a novel approach for handling the deficiencies caused by having a small and imbalanced dataset. The dataset we used is collected on Reddit, a social media platform. It contains posts and comments of individual subjects from which some of them have declared that they have been diagnosed with depression. We have used a variety of feature extractors paired up with the machine and deep learning models which at the end produced satisfactory results. It is shown that the required number of posts depends on data representation and model used in the experiment and that the proposed method for enriching small datasets works well while also outperforming other models in some cases.

1. Introduction

Depression is a common and serious medical illness that negatively affects feelings, thoughts and behaviors. By World Health Organization's[1] statistics over 300 million people are estimated to suffer from depression, equivalent to 4.4% of the world's population. However, effective treatment exists, especially in the early stages, which signifies the importance of early detection.

Resources for early detection are potentially abundant on social media. Social media platforms are becoming an integral part of people's everyday lives - the activity on them often reflects one's views and values. Therefore, the assumption is that it can provide insight into their state of mind. This implies that using this information, depression detection might be feasible, as well.

In this paper, we study how much data is needed to get satisfactory results on the depression detection task. In addition, we also present our approach to dealing with a small and imbalanced dataset. To achieve this, we train various machine learning models on a dataset of Reddit posts first presented in Losada and Crestani (2016). The dataset is described in more details in Section 3, where we also explain how the text had been processed before being fed into our models. Since we are using data from a social media platform, where subjects write in a more informal manner and often using slang expressions, text cleaning and preprocessing is a crucial step in obtaining adequate results. Furthermore, in Section 4, we explain our approach to tackling the research questions we posed earlier.

We manage to outperform our baseline model and achieve higher accuracy and F1 scores than in Losada and Crestani (2016). Evaluation process and model results are presented in Section 5. Since we think that research in this field has a lot of potential with more available data, we give our propositions for future work in detail in Section 6. In Section 7 we gather our findings and present the final conclusion.

[1] https://www.who.int/

2. Related Work

Even though it was found that depression is associated with distinctive language patterns, there is a lack of publicly available data for doing research on the interaction between language and depression. The dataset used in this paper is the first dataset for research on depression and language use, described in Losada and Crestani (2016). Authors of the mentioned paper have also defined a new measure called $ERDE$, which is based on the accuracy of the decisions and the delay in detecting positive cases. They have also provided an initial set of models which can be used as a reference for further studies. Almeida et al. (2017) suggest an approach that uses the same dataset and shows that combining information retrieval and machine learning methods gives the best results. The system that performed best is based on a multipronged approach, which combines predictions from supervised learning and information retrieval based systems. Subjects are ranked according to a similarity score based on the BM25 ranking algorithm. A neural network approach for detecting depression on social media texts is presented in Wang et al. (2018), their model combines TF-IDF and CNN classification. However, they have used a slightly bigger dataset. We can see that, although depression is a big problem in today society, research in detecting depression from language is still at an early stage.

3. Dataset

The dataset used in this paper contains textual interactions (posts or comments) from 892 subjects. 137 subjects have explicitly declared that they have been diagnosed with depression, and the remaining 755 subjects are a control group. For each subject, we have a history of writing on Reddit stored in XML files. Each writing contains:

- TITLE - title of post if available,
- DATE - date of creation,
- INFO - additional info about writing and
- TEXT - body of the text or comment.

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 1–5, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

Table 1: Dataset statistic

	Train		Test	
	Negative	Positive	Negative	Positive
Number of subjects	403	83	352	54
Average post length	23	37	27	38
Average number of posts	655	371	618	346
Average number of words	15315	14023	16722	13285
Average number of pronouns	467	588	506	527

The proposed dataset train-test split contains 486 train subjects (83 positive, 403 negative) and 406 test subjects (54 positive, 352 negative). We can see that the number of subjects is quite small and the classes are imbalanced with almost 6 times more negative than positive subjects. We will explain our approach in tackling this problem and show that for successfully training deeper models we need more data. Table 1 contains dataset statistics from where we can make a couple of interesting observations. Positive subjects are writing longer posts but they tend to write fewer posts than negative subjects. Also, they are using more pronouns in expressing themselves.

3.1. Preprocessing

Cleaning the data before being fed it into models is especially important when dealing with posts and comments on social media platforms. We used Spacy[2] for processing our data. Stop words and punctuations were removed from posts and after that word lemmas were concatenated to get clean posts or comments. We have also tested using specific parts of speech separately, but that did not produce promising results.

3.2. Dealing With Small And Imbalanced Data

Initially, all posts from a single subject were concatenated in a single document and then used as training samples. We will now propose our solution for dealing with small and imbalanced data. As stated earlier, we had 892 subjects with almost 6 time more negative subjects than positive. To solve that problem we decided to split concatenated subject posts into smaller parts which we called chunks. We have used them as individual samples. They were labeled according to the label of the respective chunk's author. Two ways of voting are proposed for aggregating the classification of individual chunks. The first is majority voting based on predicted chunk classifications, and the second one is the average of predict probabilities over each chunk, with a threshold. Chunk size and the threshold value are hyperparameters set to 250 words and 0.5, respectively. To balance out the classes when training we have decided to always include all posts of positive subjects and a fixed number of posts for a negative subject. Models are always evaluated on the whole test data.

[2]https://spacy.io/

Figure 1: System architecture

4. Our Approach

We will now present the methods we used to determine how many posts per subject are needed to attain adequate results on the depression detection task. They are applied to both representations of data that were previously described. To address the question of approximating the number of posts needed to achieve relevant results in detecting depression, we train all our models on a different number of posts per user and compare the results. For example, we start with analyzing the results produced by using only the 10 oldest posts, then we increase that number by a fixed amount and continue this sequence until a predetermined size has been reached. Finally, we compile and compare these results. Figure 1 shows general system architecture which we used in all our methods. We used a number of feature extractors paired with machine and deep learning models, described in detail in the following part of the paper.

4.1. Baseline

As a baseline, we settled on using logistic regression combined with a bag-of-words representation of the processed posts. We used an out-of-the-box implementation of these models from the scikit-learn[3] library. Bag-of-words (BOW) representation is a simple representation of text as a set of its words, along with the number of repetitions of each word. Since the size of the feature vector equals the size of the vocabulary size, training time is high. Logistic regression was combined with other approaches for feature extraction (which we describe in the next section) and has produced satisfying results, regardless of it being a relatively simple model compared to others presented in this paper. L2 penalty was chosen for regularization and class weights were adjusted to be inversely proportional to class frequencies in the input data (*balanced* mode).

4.2. Advanced Models

In this section, we describe models that we chose to compare with the baseline. Although not all of them pro-

[3]https://scikit-learn.org/stable/

duced expected results, we believe that some of them have much greater potential with a larger dataset, especially deep learning models.

4.2.1. Feature Extraction

A slightly more advanced feature extraction procedure we employed is Term Frequency-Inverse Document Frequency (TF-IDF). The idea behind this approach is that words that repeat more often in a document, but are less common across different documents, gain higher weights. Firstly, we generate the corpus by concatenating all posts of a subject in one document. Then, every document is transformed to a vector of fixed length using TF-IDF. Initial fitting of parameters is done only on training sets. A feature vector is the same size as the feature vector generated by BOW, so the training time is high as well.

However, neither of the approaches presented so far captured word semantics. Therefore, we chose to experiment with word embeddings. The first word embedding model we tried is GloVe. GloVe (Pennington et al., 2014) is a model for unsupervised learning of word representations, trained under the assumption that co-occurrence of words can be exploited for semantics (similar words appear in similar contexts). We have used pretrained GloVe word embeddings with a 100-dimensional word-embedding vector.

In addition, a pretrained word2vec model was used where words are represented as 200-dimensional vectors. These vectors are actually weights of a neural network trained to predict a word that is most probable given a context.

Since word embedding models produce fixed-size vector for every word in the document and documents (concatenated subject posts) have different lengths, a way to aggregate word semantics to represent a meaning of the document is needed. We settled on two approaches: one is a simple addition of each word representation in a document, and the other is similar, but word representations are weighted using TF-IDF coefficients.

$$subject\_vector(W) = \sum_{w \in W} word\_emb(w)$$

$$subject\_vector(W) = \sum_{w \in W} word\_emb(w) * tf\text{-}idf(w)$$

Initial experiments with BM25 ranking method and LIWC [4] features were also done, but since they did not yield promising results, these methods of feature extraction were abandoned.

4.2.2. Traditional Machine Learning Approach

Other than logistic regression, we have also tried other machine learning algorithms to find which one is the most suitable for this problem.

Support Vector Machines (SVM) achieve good performance on text classification because of their ability to generalize well in high dimensional feature spaces, due to the option of using various kernels. In this paper, the RBF kernel was chosen. Also, similar to logistic regression, class weights are set to *balanced*. We have used a grid search to optimize the C and γ hyperparameters.

We also used two ensembles. The first one is AdaBoost (Schapire, 2013), an algorithm which combines many weak classifiers, which are in our case decision trees with depth 1 (decision stumps). We have trained it with different feature extraction methods using a grid search to optimize the number of estimators and the learning rate parameter. The second one is XGBoost (Chen and Guestrin, 2016), which is an implementation of gradient boosted decision trees. Same as with AdaBoost, we optimized the number of estimators and max depth of trees through grid search.

4.2.3. Deep Learning Approach

Although the used dataset contains a low number of subjects, we chose to experiment with deep models as well. A natural approach to processing sequential data, like text in our case, are recurrent neural networks. Since our sequences are very long we have decided to use Long Short Term Memory networks (Hochreiter and Schmidhuber, 1997), or LSTMs for short. They are explicitly designed to handle the long-term dependencies and tackle the vanishing gradient problem. The first layer is the embedding layer which transforms the input according to the provided word embeddings. The second layer is 1-dimensional convolution with kernel size 1x5 used for capturing interactions between words, inspired by Wang et al. (2016). These are then used as inputs to the LSTM layer. Dropout layers are also used to prevent overfitting (Srivastava et al., 2014), which is usually a problem when the dataset is small. RMSProp was used as the optimizer during training and binary cross entropy was chosen as loss function. One thing to note is that much better results were achieved when using generated chunks, as described earlier, than using the concatenation of all subjects posts.

5. Results

To get a good estimation of evaluation measures, we have used a stratified 5-fold cross validation. Considering that all models have high accuracy because of imbalanced classes, preferable indicators for model's performance are precision, recall and F1-score.

Table 2 presents the F1-scores of the models in regards to the number of posts chosen per subject, as described earlier. To statistically compare the models which showed the best initial results, namely XGBoost with TF-IDF and voting as well as logistic regression with TF-IDF, we employ Student's t-test. With a 0.1 significance level, we can conclude that XGBoost is better for 100 posts, while logistic regression with L2 penalty always outperforms the baseline model.

If we compare results produced when using different representations, we can find that TF-IDF produces better results than the others, especially when paired with L2-regularized logistic regression. Other than that, we can notice that the performance of models for different numbers of posts depends on the representation used. For TF-IDF, F1-scores stay mostly the same after 300 posts per subject. However, when using GloVe word embedding, F1-scores are relatively high with a lower number of posts, but drop

[4] http://liwc.wpengine.com/

Table 2: F1-scores of models for different amounts of posts

Model	F1@10	F1@100	F1@200	F1@300	F1@400	F1@500	F1@600
baseline	0.5364	0.5156	0.5212	0.5693	0.5586	0.5571	0.5694
TF-IDF + Logistic Regression with L2	**0.5487**	0.5818	**0.5987**	**0.6318**	**0.6341**	**0.6305**	**0.6392**
TF-IDF + SVM	0.4686	0.5525	0.5785	0.5999	0.5737	0.6088	0.617
TF-IDF + XGBoost	0.3444	0.5399	0.5549	0.584	0.5886	0.6124	0.608
GloVe + XGBoost	0.3741	0.4401	0.4934	0.4888	0.4453	0.4613	0.4439
GloVe + AdaBoost	0.4283	0.4577	0.4804	0.4634	0.4308	0.3954	0.4336
TF-IDF * GloVe + XGBoost	0.3862	0.4902	0.483	0.4659	0.4806	0.4529	0.4569
TF-IDF * GloVe + AdaBoost	0.402	0.4895	0.466	0.478	0.4403	0.4213	0.4242
TF-IDF + LR + voting	0.5115	0.5832	0.5839	0.5471	0.5303	0.4865	0.4775
TF-IDF + XGBoost + prob average	0.5004	**0.5913**	0.5457	0.5061	0.4932	0.4278	0.3694
word2vec + LSTM + voting	0.5259	0.4255	0.1905	0.5161	0.34	0.5098	0.5

as the number of posts increase. We suspect this happens because the noise introduced by more posts outweighs the relevant semantic information they bring.

An interesting observation we can make is that the implemented voting mechanism produced significant results with an already unexpectedly low number of posts. This implies that this way of managing the issues posed by having a small and imbalanced dataset can hold its own against the more traditional approaches. In addition, if more attention is given to earlier rather than more correct detection, it may be more successful.

To compare our methods with one's described in the other papers we have trained our models on proposed dataset split, explained in the Section 3. Only two papers were found that used the same train-test split, to the best of our knowledge. It can be seen that presented models and the voting mechanism works comparably well with others, and even achieve better results in some cases, as shown in Table 3. By comparing our results with those from Paul et al. (2018) and Wang et al. (2018), we find that they achieved higher F1-scores. However, the models used in those papers are trained on a larger dataset than the one at our disposal which indicates that enlarging the dataset could bring significant improvements.

Table 3: Comparison with other approaches

Model	F1@10	F1@500	F1
TF-IDF + LR with L2	**0.562**	0.557	0.555
TF-IDF + XGBoost	0.277	**0.637**	0.597
TF-IDF + LR with L2 - voting	0.470	0.613	**0.612**
TF-IDF + LR with L1 (Losada and Crestani, 2016)	0.31	0.62	0.59
UQAMA (Almeida et al., 2017)	-	-	0.53

6. Future Work

As we have already mentioned, one of the main challenges we faced when training machine learning models for our goal was the number of subjects for whom the data had been available in the dataset. Although an increasing number of posts per subject can have conflicting effects on model performance, depending on the representation, we suspect that a greater benefit in performance can be gained from collecting data of more subjects. Therefore, to produce more definite and improved results, it is necessary to increase the size of the dataset.

In addition, our focus was less on deep models specialized for learning sequence representation and more on traditional models and ways of feature extraction used in machine learning. However, we suggest that, in future attempts, more attention is given to deep models, capable of handling long-term dependencies, such as bidirectional LSTMs. This is based on the assumption that these models will be better adjusted for extracting features across posts for a certain subject than the models presented in this paper.

7. Conclusion

We have experimented with different feature extractors and different models to find when can we, even remotely successfully, detect depression from social media posts.

Presented experiments show that the number of posts required to achieve this goal varies, depending on what type of post representation is used. TF-IDF with L2-regularized logistic regression achieves best results and stabilizes its performance after around 300 posts. GloVe word embeddings coupled with different semantic compositions give interesting results with a low number of the posts, but they degrade as number of posts increase. Also, we note that the results gained through the presented voting mechanism show that it is a valid way of tackling the problem of a low number of subjects.

References

Hayda Almeida, Antoine Briand, and Marie-Jean Meurs. 2017. Detecting early risk of depression from social media user-generated content.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system.

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

David E. Losada and Fabio Crestani. 2016. A test collection for research on depression and language use.

Sayanta Paul, Jandhyala Sree Kalyani, and Tanmay Basu. 2018. Early detection of signs of anorexia and depression over social media using effective machine learning frameworks.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation.

Robert E. Schapire. 2013. Explaining adaboost.

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.

Xingyou Wang, Weijie Jiang, and Zhiyong Luo. 2016. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2428–2437, Osaka, Japan, December. The COLING 2016 Organizing Committee.

Yu-Tseng Wang, Hen-Hsen Huang, and Hsin-Hsi Chen. 2018. A neural network approach to early risk detection of depression and anorexia on social media text.

You Think That's Funny? Humor Ranking in Tweets

Alen Carin, Tin Ivan Križ, Robert Jambrečić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{alen.carin,tin-ivan.kriz,robert.jambrecic}@fer.hr

Abstract

Humor ranking is a complex task that has not attracted much attention in natural language processing. To tackle this problem, we use a labeled dataset of funny tweets, a wide range of preprocessing methods and a variety of different models. We show that preprocessing the original tweets brings no significant improvement to the results. Additionally, our character-level convolutional neural network yields best results, achieving 70.4% accuracy on the evaluation dataset for pairwise humor comparison.

1. Introduction

Due to its subjective nature, humor ranking has always been a difficult task. Have you ever heard someone saying "*You think that's funny*" after a joke? The problem is that there are many different types of humor and people often disagree on what is funny and what is not.

Generally, there is not a lot of work done on the topic of humor ranking because of its complexity, lack of datasets and because it is not as important for the industry as some other natural language processing tasks such as emotion or plagiarism detection. In our research, we are trying to tackle the problem by comparing funny tweets on a given topic and trying to determine which one is funnier. We provide analysis of using different preprocessing methods and models. We mostly focus on character-based models because tweets mainly contain puns[1], slang, and misspelled words. Our best-performing model is showing that it can handle unbalanced data and produce state-of-the-art results for humor ranking in tweets.

2. Related Work

The task was initially introduced in (Potash et al., 2016) for the SemEval[2] competition in 2017. The authors of the paper describe their approaches to solving this task. They tried using both supervised and unsupervised approach. For the supervised approach the input to their model is a pair of tweets and the label which indicates the funnier tweet. Before training they randomly shuffled winning and losing tweets in order to have balanced class labels (approximately the same number of zeros and ones). They experimented with both word-based and character-based models and concluded that the model using character embeddings performs significantly better. Also, they made some feature experiments which didn't result in any performance gain, quite the opposite, hashtag embedding resulted in performance decrease.

We have also had great difficulties with finding worthy features and decided not to use anything except character representations for input to our models. Our approach differs from others in the way that we don't feed the model with tweet pairs, but rather with tweets and their class labels (0, 1 or 2). Then, we predict class probabilities for each tweet and compare zero class probabilities between tweets that are labeled differently. This approach is much simpler and has proven to perform significantly better on the task.

3. Dataset

Our dataset is publicly available[3] and originates from a TV show "@midnight"[4] where contestants get points for funny answers. One segment of the show is "#HashtagWars" where contestants get a topic (e.g. "#VegasMovies") and they have to make funny examples of the topic (e.g. "A Fistful of Quarters"). The show then encourages the audience to tweet their own respones on the topic and publishes the top 10 funniest responses.

Our dataset contains a training set with 101 topics and a evaluation set with 6 additional topics. Each topic contains between 12 and 182 tweets for a total of 11,280 tweets. Each of the tweets is labeled as "0" (not top 10), "1" (top 10) or "2" (the funniest). Tweets are hard to train on because of their length. Their length ranges between 22 and 430 with an average of 58.4 characters per tweet. This is because they are mainly just short puns written as responses to the given topic.

All the tweets in the dataset contain the show mention "@midnight" and the topic hashtag (e.g. "#VegasMovies"). They may also contain any of the other Twitter-specific content (hereafter referred to as "Twitter specifics") such as picture URLs, other hashtags, other mentions, smileys and emojis.

4. Experimental Setup

We include two types of analyses. The first type focuses on how preprocessing of Twitter specifics influences our humor ranking. The second part includes analysis of different model performance on the original dataset.

[1] A joke exploiting the different possible meanings of a word.

[2] http://alt.qcri.org/semeval2017/task6/

[3] http://alt.qcri.org/semeval2017/task6/index.php?id=data-and-tools

[4] http://www.cc.com/shows/-midnight

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 6–9, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

4.1. Preprocessing

The tweets in our dataset contain a lot of Twitter specifics such as hashtags, mentions, URLs, emojis and smileys. On one hand, this content is expected to represent noise in our tweets because it is not related to the actual joke. On the other hand, this extra content could bring some valuable information to our models.

In this section, we investigate the impact of preprocessing Twitter specifics from input tweets on the humor ranking task. We are using Twitter preprocessor library[5] to differently preprocess data and character-level XGBoost model with 80 character padding and max depth set to 5 where every character is represented as a single-dimension unique number. This setup provides good results and is very fast to train, enabling us to try a wide range of different preprocessing methods.

4.2. Models

There are quite a few models we have tried in order to solve this task. As a baseline we consider random guessing which for this task is 50% due to the pairwise comparison evaluation method we used which will further be described in the evaluation section. Because of the complexity of this problem it is not an easy task to achieve much greater results than the random-guessing baseline.

As input to our model we experimented with various types of tweet representations. Our first approach was to represent each word in a tweet with pre-trained GloVe vectors Pennington et al. (2014) which have been trained on 2B tweets and contain 1.2M word vocabulary. After every word is converted into a 25 dimension vector, we would add them element-wise to acquire tweet representation. However, we noticed that there are a lot of out-of-vocabulary words which is most of the time a consequence of users concatenating multiple words together or using colloquial language. This suggested we should use character embeddings as input to our model instead of word embeddings.

For character-based models our vocabulary consists of 96 different characters, plus "<UNK>" tag for unknown characters. There were three different representations of characters we used:

- single dimension unique numbers
- one-hot vectors
- pretrained character embeddings derived from the GloVe 840B/300D dataset[6].

To start with, we experimented with some shallow classification models such as Logistic Regression and Support Vector Machines. We became aware that different models tend to work better on different topics, so we decided that we should use an ensemble of multiple models. We used XGBoost, introduced in Chen and Guestrin (2016), because of its speed and overall good performance. XGBoost stands for extreme gradient boosting and uses gradient boosted decision trees. The model is very popular among the Kaggle community where it is used for a large number of competitions.

Finally, we implemented deep neural models using the Keras library (Chollet and others, 2015) which is running on top of TensorFlow. Our best results were achived with character-based CNN. Our model's setup follows that of Zhang et al. (2015). It is a 14 layer-deep model which consists of embedding layer, two convolutional layers both followed by activation and max-pooling layers. The output of convolutional layers is flattened and fed into two fully connected layers with 256 neurons. Softmax function is applied to the output of our model and the results are probabilities for each class of the task. The training objective was to minimize categorical loss and the optimization was performed using the Adam optimizer from Kingma and Ba (2014).

5. Evaluation

To validate different model's performance, we are using leave-one-out cross-validation method on the train set (101 topics). In other words, in each iteration we are training our model on 100 topics and then validating it on the remaining one. We decided to compare different models on the training set because it is effectively testing the model 101 times, whereas it would be tested only 6 times on the evaluation set.

We have further evaluated our best-performing model on the evaluation set (6 topics) and compared the results with the results of other teams on SemEval competition Potash et al. (2017).

Performance on a single topic is validated as probability that our model would be able to tell which of the two given tweets is labeled as funnier. This probability is acquired by testing all combination of pairs between tweets with different labels which is approximately 1,000 pairs per topic. We capture the accuracy on each topic separately and calculate the micro-average[7] accuracy across all topics in evaluation set. This process enabled us to choose appropriate parameters for the model.

6. Results

6.1. Preprocessing

In this section, we will introduce results of how different ways of preprocessing tweets influences character-level XGBoost model performance. Table 2 shows that no preprocessing method proved notable increase in performance in comparison with using raw data. Even though we have an increase on some preprocessing methods, further research has proved that it is only due to the choice of the model and its parameters. In conclusion, learning with full preprocessing (all Twitter specifics) yields significantly[8] worse results than with raw data. Is the Twitter-specific content in tweets containing useful information for humor ranking?

Table 3 shows that tweets are less probable to be awarded a place in the top 10 funniest tweets if they contain any additional Twitter-specific contents. The biggest difference is with the tweets containing additional mentions (e.g.

[5] https://github.com/s/preprocessor

[6] https://github.com/minimaxir/char-embeddings

[7] Average across all the topics weighted by the topic sizes.

[8] With p-value less than 10^6 using permutation testing.

Table 1: Our model compared to official results from SemEval competition provided in (Potash et al., 2017). In the table "Bad Job" corresponds to the hashtag BadJobIn5Words, "Break Up" corresponds to BreakUpIn5Words, "Broadway" to BroadwayACeleb, "Cereal" to CerealSongs, "Shakespeare" to ModernShakespeare, and "Christmas" to RuinAChristmasMovie.

Team	Bad job	Break Up	Broadway	Cereal	Shakespeare	Christmas	Acc Micro Avg
TARgedy (that's us)	0.698	**0.822**	0.689	0.516	0.755	**0.685**	**0.704** ($\pm$0.102)
HumorHawk	**0.704**	0.723	0.643	0.492	**0.789**	0.673	0.675 ($\pm$0.101)
TakeLab	0.641	0.576	**0.716**	**0.704**	0.543	0.683	0.641 ($\pm$0.071)
HumorHawk	0.603	0.620	0.627	0.588	0.726	0.650	0.637 ($\pm$0.049)

Table 2: Results with different types of preprocessing methods. #topic and @midnight refer to the topic hashtag (e.g. "#PrisonBooks") and the show mention which are present in all tweets in the dataset. Average accuracy shown is computed as micro-average accuracy over the dataset.

Removing	Avg Acc	Min Acc	Max Acc
Nothing	0.6538	0.3714	0.8140
Everything	0.5529	0.114	0.7809
All #Hashtags	0.6442	0.3429	0.8615
All @Mentions	0.5877	0.2970	0.8057
URLs	0.6573	0.400	**0.8731**
Emojis, Smileys	0.6621	0.400	0.8300
@midnight	0.6545	**0.4285**	0.8727
#topic	**0.6669**	0.3143	0.8723
@midnight, #topic	0.5678	0.1714	0.7923

Table 3: Statistics on average grades (labels) depending on positive (POS)/negative (NEG) presence of Twitter specifics and number of such samples. Since all the tweets contain the show mention and the topic hashtag, they are not considered for the purpose of this result. For comparison, average grade across all the dataset is 0.0877.

Contains	Samples		Avg grade	
	Pos	Neg	Pos	Neg
Other Hashtags	1417	9859	0.0473	0.0935
Other Mentions	555	10721	0.0126	0.0916
URL	384	10892	0.0260	0.090
Emoji or Smiley	19	11257	0.053	0.088

"@SomeName") having 7.3 times lower probability of being awarded a place in the top 10 funniest tweets than the ones who don't contain them. This result proves that there is a lot of information in knowing whether a tweet contains some Twitter-specific content or not. For this result, we have ignored the usual topic hashtag (#TopicName) and show mention ("@midnight") because they are present in all the tweets.

Additionally, we wanted to get the most out of both approaches with tokenization. By removing the show mention and topic hashtag and tokenizing the remaining Twitter specifics with unique characters, we would still have the information about the presence of Twitter specifics, but wouldn't allow its very content to negatively influence our character-level model. However, our further investigation showed that tokenizing Twitter-specific content makes the model perform significantly worse (-7.9%) than training on raw data. The reason lies in the fact that we can't exploit the knowledge from Table 3 without removing the topic hashtag and show mention (which are present in all tweets). While removing just one or the other actually slightly improves model results (see bottom of Table 2), removing both of them results in significant loss of accuracy on the dataset (-8.6%).

We believe that the reason of this phenomenon is that our tweets are already small to begin with and any additional preprocessing makes them even smaller which consequently makes our model's prediction more susceptible to random guessing.

Table 4: Accuracy of Logistic Regression (LR), XGBoost (XGB) and Convolutional Neural Network (CNN) model evaluated with the leave-one-out method on train set.

Model	Min Acc	Max Acc	Acc Micro Avg
LR	0.3788	**0.9384**	0.6029
XGB	0.3714	0.8533	0.6597
CNN	**0.4853**	0.8651	**0.6831**

6.2. Models

Firstly, Table 4 shows the results of performance of character-level Logistic Regression (LR), XGBoost (XGB) and Convolutional Neural Network (CNN) models. The CNN proved to perform significantly better than all the other models.

We further evaluated our best-performing model (character-level CNN) on the evaluation set and averaged the results of 10 consecutive runs. The results on each topic as well as the average is shown in Table 1. It shows our results on each topic independently as well as the micro-average accuracy across all topics compared with the official results on SemEval competition for this task. Our model (the first one in the table) achieved best results on two topics and has the best overall result (70.4%).

7. Conclusion

In this paper, we have showed that preprocessing data for the humor ranking task brings no statistical significance to performance as the tweets are already small to train character-level models on.

Furthermore, we show that the best-performing model on the task is a character-level CNN that is able to determine the funnier of any two tweets on a given topic with a 70.4% probability on the evaluation dataset. This result is currently the highest for the humor ranking task and represents state-of-the-art performance.

Nevertheless, we see room for further improvement. Firstly, it would be interesting to implement some external knowledge since a lot of the puns are related to the actual movies, people or events. Also, including features concerning Twitter specifics could be beneficiary for our model's performance since tweets without them have higher probability to be awarded a place in the top 10 funniest.

References

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

François Chollet et al. 2015. Keras. https://keras.io.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, Dec.

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, Doha, Qatar, October. Association for Computational Linguistics.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2016. #hashtagwars: Learning a sense of humor. *CoRR*, abs/1612.03216.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2017. SemEval-2017 task 6: #HashtagWars: Learning a sense of humor. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Vancouver, Canada, August. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. *arXiv e-prints*, page arXiv:1509.01626, Sep.

Redefining Cancer Treatment: Comparison of Word2vec Embeddings Using Deep BiLSTM Classification Model

Tin Ceraj, Ivan Kliman, Mateo Kutnjak

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{tin.ceraj,ivan.kliman,mateo.kutnjak}@fer.hr

Abstract

When dealing with genetic mutations, medical expertise is needed for the analysis of vast amounts of gathered knowledge. In this paper, we compare different Word2vec embedding models for deep learning classification of genetic mutations that can affect tumor growth and thus impact patient's health. Embedding models are generated from specialized and general corpora with and without preprocessing of medical descriptions. Embedding quality comparison is done with a multilayer bidirectional LSTM classification model.

1. Introduction

Unstructured textual data still contains most of the human knowledge. Methods like *Word2vec* represent natural language words in multidimensional space and provide relations between them depending on their context. Encoding words into real value vectors preserves information about target word and context and simultaneously crams variable-length sequences to fixed-length vectors. But, like any machine learning model, *Word2vec* contains the knowledge of the corpora training it was performed on.

As medical texts include the most specialized information comprehended only by field experts, we decided to develop a model capable of understanding its complex and specific structure. We applied to Kaggle's competition "Redefining Cancer Treatment" where medical entries describe genetic mutations and competitors are required to classify given mutations. Currently, the interpretation of provided mutations is being done manually by experts and demands lots of time and resources.

In this work we compare *Word2vec* embeddings trained on common knowledge datasets such as *Wikipedia*, *Google News*, *English CoNLL17 Corpus* and custom models trained on domain-specific data. We also compared text preprocessing on training data. Finally, embeddings are used as input to deep learning multilayer LSTM model used for the classification of genes mutations.

Our results show that the usage of specially trained *Word2vec* models on domain-specific data is crucial for the effectiveness of classification and heavy preprocessing decreases the number of correct predictions.

2. Related work

Since *Word2vec* has been introduced (Mikolov et al., 2013), word representation as vectors has been used for tasks like semantic similarity. *Word2vec* has also become quite popular as embedding for deep learning models in various semantic (Yu and Dredze, 2014) and information retrieval problems (Ganguly et al., 2015). Two types of *Word2vec* models exist: Continuous-bag-of-words predicts target word from given context while Skip-gram predicts context from the current word. Figure 1 shows both variations.

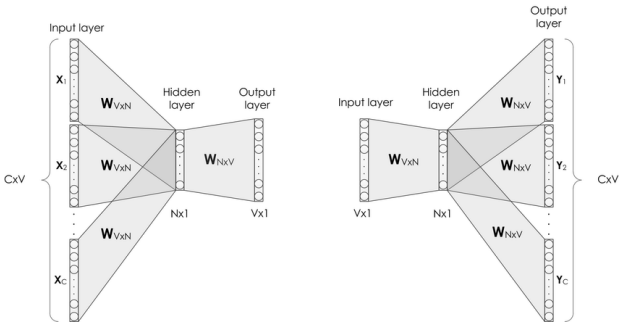


Figure 1: Continuous-bag-of-words (CBOW) and Skip-gram *Word2vec* shallow models trained to reconstruct linguistic context of words.

Deep learning has also progressed from a simple LSTM network (Hochreiter and Schmidhuber, 1997) to complicated multilayered and bidirectional models (Graves et al., 2005), being able to better represent complex textual sequences.

Popular approach is combining *Word2vec* encodings as input to other machine learning models for classification tasks such as *support vector machines* (Lilleberg et al., 2015) or deep models such as convolutional network models (Mandelbaum and Shalev, 2016).

Sahu et al. (2016) proposed automated feature extraction from clinical text with convolutional networks to remove the dependency of domain expertise for specific tasks.

We analyze how different settings of *Word2vec* embeddings affect the gene mutations classification based on highly specialized medical descriptions while using state-of-the-art deep learning bidirectional LSTM model.

3. Dataset

Genetic mutations descriptions and classification data were provided by Kaggle[1] as a part of its competition "Redefining Cancer Treatment" for automated classification of genetic mutation with regard to their impact on cancer growth.

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

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 10–13, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

Table 1: Features of pre-trained *Word2vec* models

Model	Type	Vector size	Window size	Lemmatization	Vocab size
Google News 2013	Skip-gram	300	5	False	2883863
English Wikipedia Dump of February 2017	Skip-gram	300	5	True	296630
English CoNLL17 corpus	Skip-gram	100	10	True	4027169

The entire dataset is split into training and test set. The original training set consists of 3321 annotated examples with a textual description of genetic mutation, gene, variation, and nine affiliated classes, while the test set contains 5668 examples without genetic mutation class given.

The test set contains a total of 5668 examples that include 5300 machine-generated ones. The test set that Kaggle uses for ranking (named *stage-2*) has 986 examples of *real* data that is split to private and public subsets that contain 76% and 24% of data respectively.

Datasets named "train" and "test" provided by *Kaggle* are merged into one set for word embeddings and were later split to training set (90%) and validation set (10%). For deep classification model training only the "train" set was used because it is the only one that contains the labeled data. Dataset named "stage-2" is used for testing as it needs to be uploaded to *Kaggle* for competition ranking.

4. Approach

4.1. Embeddings

A comparison between *Word2vec* embeddings is done with respect to several factors. Firstly, we tested how embeddings pre-trained on English corpora behave in comparison with the custom trained model. All pre-trained models are created by Fares et al. (2017) and are free to download [2]. Pre-trained models were trained on Google News 2013, English Wikipedia Dump of February 2017 and English CoNLL17 corpus datasets and have differences in hyperparameters like context window size and vector dimensionality. The most important hyperparameters are shown in Table 1.

The second variation that is tested is the effectiveness of preprocessing on classification. Preprocessing methods contain the removal of punctuation, removal of symbols, removal of stop words, removal of tokens which consist only of digits and finally lemmatization.

Gene and variation did not affect the *Word2vec* embedding model and were not used in word encoding. Each of the custom models has the same hyperparameters (window size, vector dimensionality, negative samples, network architecture type) as their associated pre-trained counterpart.

4.2. Model

Two baseline models are given. One randomly labels example with distribution equal to the distribution of classes in the labeled training dataset. That means that the most common class in the training set is represented the most in randomly chosen labels. The second baseline simply labels every example with the most represented class in the training set.

[2] http://vectors.nlpl.eu

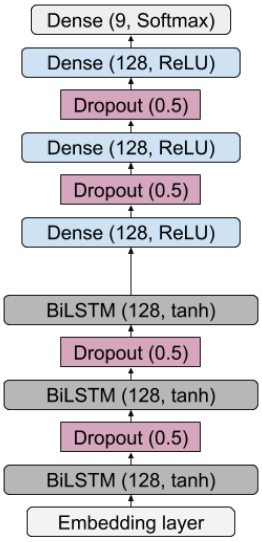


Figure 2: Deep learning classification model with embedding layer as input, three bidirectional LSTM layers with 128 units, three dense layers with 128 units, dropout with 0.5 rate and output layer with 9 probabilities for each class

Word embeddings generated with variations in hyperparameters are fed to the deep model which is used for the final classification and comparison of word encoding models. Besides the difference in hyperparameters, the classification model is constant through experiments.

The model which is used is bidirectional LSTM and it consists of dense, LSTM and dropout layers, as shown in Figure 2. It uses *Word2vec* embeddings as input layer and results with a probability for each of 9 classes for genetic mutations.

The optimizer used in this model is *Adam* (Kingma and Ba, 2014) with a learning rate set to 0.0001. We trained our model with a batch size of 300 over 50 epochs. We used validation loss as a monitor. That means the model is saved only if it has better validation loss than the previously saved model (from the epochs before). All of the mentioned hyperparameters were selected with empirical experimentation.

5. Results and Evaluation

Datasets "train" and "test" are merged and split to training set containing 90% of examples and validation set contain-

Table 2: Multi class log losses of different *Word2vec* embeddings

Model	Validation loss	Public score	Private score
Baseline (most common class)	-	25.552	33.157
Baseline (classes distribution)	-	30.662	29.846
Google News	1.307	1.498	3.909
Our Google News	**1.292**	1.368	3.578
Google News (preprocessed)	1.348	1.449	**3.547**
Our Google News (preprocessed)	1.333	**1.311**	3.850
Wikipedia Dump	1.432	1.679	3.422
Our Wikipedia Dump	**1.282**	1.412	**3.207**
Wikipedia Dump (preprocessed)	1.477	1.524	3.260
Our Wikipedia Dump (preprocessed)	1.380	**1.360**	3.647
CoNLL17 Corpus	1.324	1.479	3.583
Our CoNLL17 Corpus	**1.276**	1.428	3.449
CoNLL17 Corpus (preprocessed)	1.415	1.462	**3.434**
Our CoNLL17 Corpus (preprocessed)	1.369	**1.357**	3.479

ing 10% of examples. Cross-validation was used to test the effectiveness of embedding models. Testing was done with *Kaggle's* dataset named "stage-2" because it is used for leaderboard ranking submission. Submission of predicted labels for test dataset results in public and private score. Dataset "stage-2" provided by *Kaggle* does not contain all labels and the evaluation through the *Kaggle* website is required. Results are shown in Table 2.

The effectiveness of *Word2vec* models with variants of pre-processing and pretraining are shown in Table 2.The models in the table with the prefix "our" are trained by us, while the rest of the name shows which pre-trained model was used as a representative of general word embeddings. Public and private scores represent **multi-class log losses** calculated by *Kaggle* when uploading the prediction for each of the test examples.

All of the pre-trained embeddings and custom embeddings managed to outperform baseline models by a large margin.

Models that achieved the lowest cross-validation losses are all results from a custom trained *Word2vec* model trained on medical descriptions without preprocessing. As for public scores, custom models have the lowest multi-class log losses with all *Word2vec* source categories where datasets are preprocessed. Private scores show that the most appropriate model for classification is a pre-trained *Word2vec* model that works with the preprocessed dataset. It should be mentioned that a custom trained model that does not work with preprocessed data is close by score.

When comparing datasets for each score validation, for pre-trained models vocabulary size is the biggest factor in *Word2vec* model accuracy. Here the pre-trained *Wikipedia Dump 2017* performs the worst.

When comparing all datasets, the lowest cumulative loss is achieved by the custom trained embedding models on a non-preprocessed dataset. We believe this is due to the fact that the medical data is highly sensitive to strong processing. Special nomenclature and abbreviations that are common in specialized medical descriptions have been lost or their meaning has been changed in preprocessing and cannot be vectorized properly with the pre-trained model trained on general texts understandable to the majority of people.

6. Discussion

Relatively small training set provided by *Kaggle* is sufficient for competition purposes but a larger dataset is needed for developing a professional tool for gene mutation classification. The limited dataset provided by *Kaggle* hardly exceeds 700 MB. Annotation of greater datasets also requires work by experts which is more costly and time-consuming.

Experimentation with different deep learning models may be conducted. Deeper models with convolutional layers could improve results. Hyperparameters should also be chosen with a thorough space search.

Additional variations of preprocessing need to be done. Specifically, we suggest using preprocessing techniques such as switching text to lowercase, stemming and stop words removal which we believe do not omit information as digit tokens removal does.

7. Conclusion

In this paper, we have compared several different pre-trained *Word2vec* embeddings for deep learning classification with matching *Word2vec* models trained on the specialized medical dataset with variations, such as preprocessing of genetic mutations descriptions.

Evaluated results show that the most promising results are got from the limited dataset on custom embeddings and further improvements are possible with a larger dataset corpora, more complex models and longer training periods.

References

Murhaf Fares, Andrey Kutuzov, Stephan Oepen, and Erik Velldal. 2017. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *NODALIDA*.

Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth J. F. Jones. 2015. Word embedding based generalized language model for information retrieval. In *SIGIR*.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2005. Bidirectional lstm networks for improved phoneme classification and recognition. In Włodzisław Duch, Janusz Kacprzyk, Erkki Oja, and Sławomir Zadrożny, editors, *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*, pages 799–804, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

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

J. Lilleberg, Y. Zhu, and Y. Zhang. 2015. Support vector machines and word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*, pages 136–140, July.

Amit Mandelbaum and Adi Shalev. 2016. Word embeddings and their use in sentence classification tasks. *CoRR*, abs/1610.08229.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Sunil Kumar Sahu, Ashish Anand, Krishnadev Oruganty, and Mahanandeeshwar Gattu. 2016. Relation extraction from clinical texts using domain invariant convolutional neural network. *CoRR*, abs/1606.09370.

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *ACL*.

How Much Implicit Knowledge Is There in Deep Learning Models?

Ivan Crnomarković, Mihovil Ilakovac, Roman Kerčmar

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ivan.crnomarkovic, mihovil.ilakovac, roman.kercmar}@fer.hr

Abstract

Model interpretability is an important topic when considering deep learning models. It is beneficial to understand how models work and what are their limitations. Results of the "Character Identification on Multiparty Dialogues" task on SemEval 2018 have shown that it is possible to build models with relatively good performance on that task, but with those models being deep learning models, we have little understanding of how much knowledge they really have. In this paper, we analyze the winning model on the task and examine how much knowledge about personal relationships it has by testing it on handcrafted sentences. Furthermore, we build our own rule-based model and observe that it outperforms the winning model on the SemEval task considerably.

1. Introduction

Since the evolution of Web 2.0 amount of user-generated content is growing exponentially. A huge amount of that content is made of multiparty dialogues. Due to the nature of multiparty dialogues where several speakers take turns to complete a context, character identification is a crucial step for adapting higher-end NLP tasks (e.g., summarization, question answering, machine translation, etc.) to this genre (Choi and Chen, 2018).

Character identification is a combination of entity linking and coreference resolution. It requires connecting mentions (she, he, him, ...) to explicit entities (Joey, Ross, ...) in text and then connecting those entities to database entities.

The task is challenging because it requires an understanding of colloquial writing, slangs, sarcasm, rhetorical questions and also knowledge of interpersonal relationships.

SemEval-2018 Task 4 (Choi and Chen, 2018) was the latest shared attempt to enhance the state of the art on this task and it was dominated by various deep models. A lot of effort is invested in building better and better models with the goal to enhance the state of the art results, but not much in exploring how much knowledge these models contain.

Because the understanding of interpersonal relationships is important for this task, our main goal is to test how much of that knowledge deep models gather during training. Testing is conducted with the winning winner of SemEval-2018 Task 4, the model created by AMORE-UPF team (Aina et al., 2018) [1]. To test this we compare it against the rule-based model on character identification task to see how much can be achieved without any implicit knowledge at all. Then in a series of tests, we then analyze in-depth model's knowledge.

Firstly, in the following section, we review work related to deep learning model analysis. In sections Task Description and Dataset we describe the task and the dataset, respectively. Then in section AMORE-UPF Model we describe AMORE-UPF model and after it in section Rule-based Model rule-based model. In-depth testing is done in sections Results and Knowledge Testing.

[1] https://www.upf.edu/web/amore

2. Related Work

There is work done on the comparison of the models that applied to the SemEval task (Choi and Chen, 2018) but they analyze the performance of the models that were made and applied to the task in its first iteration, one of them being AMORE-UPF model (Aina et al., 2018). We create a new model and compare the performances of previous ones with it.

Some work is also done in the domain of trying to understand what those models learn (Aina et al., 2019). They are mostly done by analyzing the hidden layers of the models and seeing if there is structure in it. They also analyze how they do on lower frequency entities and conclude that the models presented in the task make poor use of linguistic context and also show the need for model analysis (Aina et al., 2019). We try to incorporate more linguistic context in our model and also do model analysis on tailored examples.

3. Task Description

Given a dialogue transcribed in the text the task is to find the entity for each mention. Mention can be either active or passive in the dialogue. An entity is an actual person that the mention refers to. In Figure 1, entities such as Ross, Monica, and Joey are the active speakers of the dialogue, whereas Jack and Judy are not although they are passively mentioned as "mom" and "dad" in this context.

4. Dataset

Dataset is a corpus of transcripts from popular TV show Friends[2] and it is provided for SemEval 2018 Task 4. Transcripts cover the first two seasons of the show. Each season is split into episodes, episodes into scenes and scenes into tokens. Most of the corpus is made of dialogues where topics vary greatly, language is informal, full of humor, sarcasm, and metaphors. We can say that this corpus realistically mimics daily conversations.

The dataset contains 47 episodes with 374 scenes for the training set. There are 13 280 mentions of 372 different entities in it. For the evaluation, there are 7 episodes with 74 scenes that contain 2429 mentions of 106 entities.

[2] https://en.wikipedia.org/wiki/Friends

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 14–17, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

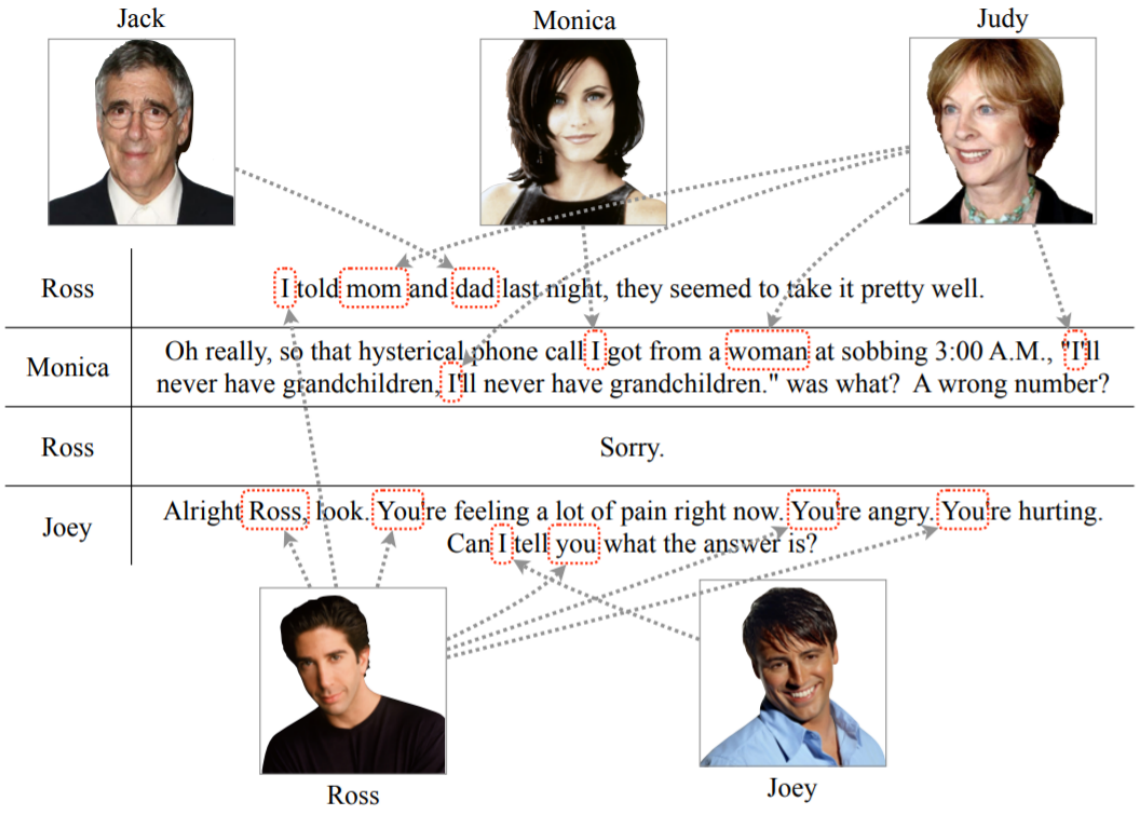


Figure 1: An example of character identification, excerpted from the Season 1 Episode 1 of Friends, where mentions are indicated in red boxes and entities are linked by arrows.

Tokens in the dataset acompanied with a season and episode annotatation, scene number, token number, and raw token. Lemmatization, part-of-speech tagging, parsing, and detection are already conducted and each mention is assigned a unique character identifier. Each token has a label whether the token is a character mention so the system can only output labels for those tokens. Example of dataset annotation is visible in Figure 2.

5. Models

5.1. AMORE-UPF Model

AMORE-UPF model (Aina et al., 2018) won the SemEval competition task of "Character Identification on Multiparty Dialogues" in 2018 which used the dataset and the task that is also used in our research.

Model is comprised of a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) with explicit modeling of the entity library which stores the representations of entities. Besides that, the model does not use any features except the speakers and words, like POS tags, lemmas, etc. In Figure 3 we can see the structure of the model.

The authors test the model against the regular bidirectional LSTM model and show that the entity library yields significant improvement on the set of all entities, not just the main ones where many of them are not mentioned a lot of times in the data.

5.2. Rule-based Model

Our model is built using five very simple rules and it is built in such a way that it does not contain any explicit or implicit interpersonal knowledge. The purpose of this model is to see what results we can achieve using just a few simple rules and to compare that against the results of deep models. That comparison gives us an approximation of how much knowledge deep models actually contain. Rules are as follows:

- RULE 1 - if there is just a name or surname we connect it to the full name (for example if there is "Al" we connect it to "Al Pacino"),
- RULE 2 - if a lemma is equal to "I" or "my" we say that target entity is the current speaker,
- RULE 3 - if a lemma is equal to "you", "dear", "sweetie", etc. we say that target entity is the speaker that spoke before the current speaker,
- RULE 4 - if a lemma is equal to "guy", "friend", "person", etc. and "you" appeared inside last four words we say that target entity is the speaker that spoke before the current speaker,
- RULE 5 - if a lemma is a third-person pronoun or equal to "guy", "friend", "person", etc. and explicit name appeared inside the last 80 words we say that target entity is that explicit name.

```
s1e1u38  0   0      I   PRP  (TOP(S(S(NP*)      I   -   -  Ross       *     (7)
s1e1u38  0   1   told   VBD          (VP*   tell   -   -  Ross       *      -
s1e1u38  0   2    mom    NN          (NP*    mom   -   -  Ross       *     (9)
s1e1u38  0   3    and    CC            *     and   -   -  Ross       *      -
s1e1u38  0   4    dad    NN            *)    dad   -   -  Ross       *    (10)
s1e1u38  0   5   last    JJ      (NP-TMP*   last   -   -  Ross  (TIME*      -
s1e1u38  0   6  night    NN          *)))  night   -   -  Ross       *)     -
s1e1u38  0   7      ,     ,            *      ,   -   -  Ross       *      -
s1e1u38  0   8   they   PRP         (NP*)   they   -   -  Ross       *      -
s1e1u38  0   9  seemed   VBD         (VP*   seem   -   -  Ross       *      -
s1e1u38  0  10     to    TO       (S(VP*     to   -   -  Ross       *      -
s1e1u38  0  11   take    VB         (VP*   take   -   -  Ross       *      -
s1e1u38  0  12     it   PRP         (NP*)     it   -   -  Ross       *      -
s1e1u38  0  13 pretty    RB       (ADVP* pretty   -   -  Ross       *      -
s1e1u38  0  14   well    RB       *)))))   well   -   -  Ross       *      -
s1e1u38  0  15      .     .         *))      .   -   -  Ross       *      -
```

Figure 2: Example of the dataset annotation

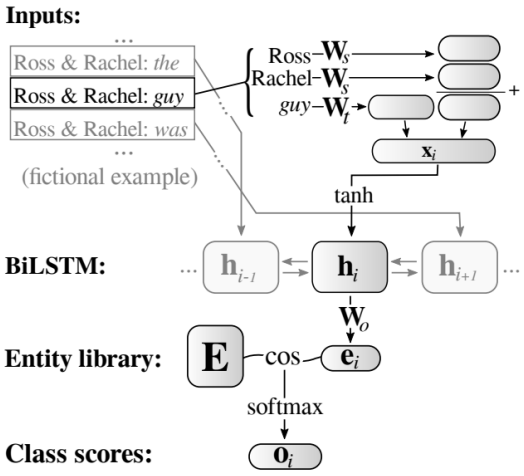


Figure 3: The AMORE-UPF model (bias not depicted). (Aina et al., 2018)

If no rule is satisfied we say that target entity is the most common entity.[3] We kept it small and simple because even this simple model gives comparable results, as we show in section Results and Knowledge Testing

6. Results and Knowledge Testing

Firstly, we compare the results of the rule-based and AMORE-UPF model on the character identification task. Results give us a clue about how good are the current deep models on this task and how much implicit knowledge they contain. Then we examine that knowledge in a series of tests. Sentences are constructed in such a way that the speaker mentions some entity with whom he has some relation (friend, spouse, parent, etc.). Constructed this way these sentences test interpersonal relationships knowledge of the model.

Table 1: Results of AMORE-UPF model and rule-based model. MC stands for most common.

Model	Results in %			
	All entities		Main Entities	
	acc	f1	acc	f1
AMORE-UPF	**74.7**	41.1	**77.2**	**79.4**
MC	18.9	2.7	18.9	16.3
MC + rule 1	22.6	22.0	22.6	27.9
MC + rule 1,2	56.1	46.5	56.1	69.5
MC + rule 1,2,3	69.8	55.4	69.8	77.1
MC + rule 1,2,3,4,5	72.7	**62.6**	72.7	78.1

6.1. Results

Results are shown in Table 1. All tests are done on the official test set. We use accuracy and F1-measure (macro) to evaluate the models. We can see that the rule-based model yields results comparable to the results of the AMORE-UPF model. Even more surprising are the results of F1 measure on all entities where the rule-based model gives better results. This would suggest that the AMORE-UPF model does not contain much implicit knowledge, for example, understanding of interpersonal relationships.

6.2. Knowledge Testing

Before we start interpreting the results it is important to notice that the model assigns an entity to each input word. Because mention detection (section Dataset) is already done and because we need to connect entities only to mentions the last step would be just to filter those outputs that come from words labeled as mentions. We kept the whole output because we think that gives a better picture of the model's knowledge.

Tests are divided into two groups. Examples of first group tests are given in Figure 4. Their purpose is to test how well the model recognizes the feelings of one character

[3] The most common entity in the training set is *Ross Geller*.

```
Joey Tribbiani: I     (Joey Tribbiani)
Joey Tribbiani: hate     (Joey Tribbiani)
Joey Tribbiani: you     (Monica Geller)

Joey Tribbiani: I     (Joey Tribbiani)
Joey Tribbiani: love     (Joey Tribbiani)
Joey Tribbiani: you     (Monica Geller)
```

Figure 4: First group tests examples. Entity before ":" is the speaker and the entity in the brackets is suggested by the model.

towards other characters. Every sentence starts with a first-person pronoun that marks the speaker that expresses some feeling (love, like, hate, etc.) toward another character that is marked with third person pronoun, the goal is to predict the entity of that third person pronoun. As we can see every test has more than one correct answer, so we considered it to be a very easy test.

Model passed four out of nine tests, but if we look at Figure 4 we can see that it does not have impressive knowledge. Results like this, when the model said that the same speaker has different feelings for the same entity in different sentences, repeated through tests.

An example of second group tests is visible in Figure 5. Their purpose is to test knowledge of relationships (friend, ex-wife, sister, husband, etc.) between characters. Sentences are written in such a way that they always contain a third person pronoun or a noun (that is not a name) that marks exactly one entity, the goal is to guess that entity. As we can see there is always only one correct answer, so we considered this to be a much harder test.

Model passed four out of eight tests, but if we look at Figure 5 we can see the same thing as with previous tests. In the first sentence, the model said that Ross Geller's ex-wife is Ross Geller. Words "she" and "sister" in the second sentence should be labeled with the same entity but that it is not the case. It is interesting how model guessed correctly entity for the word "sister" and despite that failed on the word "she".

We tried to keep test sentences as simple and as short as possible, also not a single one contained sarcasm, metaphors or required understanding of entailment. Also, most target entities were main characters that appeared in most scenes so consequently their relationships with other characters were often visible. Despite all of that model did not perform brilliantly on the tasks of identifying feelings and interpersonal relationships. In the future, we would like to dig even deeper into the model and try to understand why it sometimes hits and sometimes misses. We would also like to try to explore other models that model interpersonal relationships explicitly.

7. Conclusion

Our work is related to the problem of interpretation of black-box models and this was an attempt to shine some light on the inner-workings of the AMORE-UPF model. Using the task of "Character Identification on Multiparty

```
Ross Geller: I     (Ross Geller)
Ross Geller: saw     (Ross Geller)
Ross Geller: my     (Ross Geller)
Ross Geller: ex-wife     (Ross Geller)

Ross Geller: She     (Carol Willick)
Ross Geller: is     (Rachel Green)
Ross Geller: my     (Ross Geller)
Ross Geller: younger     (Phoebe Buffay)
Ross Geller: sister     (Monica Geller)
```

Figure 5: Second group tests examples. Entity before ":" is the speaker and the entity in the brackets is suggested by the model.

Dialogues" to probe a deep learning model proved to be insightful. The deep learning model seemed to guess the correct answer rather than actually contain proper interpersonal relationships knowledge. We can conclude that the AMORE-UPF model designed for a specific task has little implicit knowledge which can be used for other applications. More specifically, the model used for entity identification cannot be used for determining interpersonal relationships. This implies that models should be fine-tuned for a specific task and offer little flexibility outside their specific task. We think it is important to develop a better understanding of deep learning models and their capabilities. Users should always adjust their expectations when considering the power of such models. We believe that incorporating more knowledge about interpersonal relationships between entities would improve the models' score on the SemEval task.

References

L. Aina, C. Silberer, I. T. Sorodoc, M. Westera, and G. Boleda. 2018. Amore-upf at semeval-2018 task 4: Bilstm with entity library. In *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*, pages 65–69.

Laura Aina, Carina Silberer, Matthijs Westera, Ionut-Teodor Sorodoc, and Gemma Boleda. 2019. What do entity-centric models learn? insights from entity linking in multi-party dialogue. *ArXiv*, abs/1905.06649.

J. D. Choi and H. Y. Chen. 2018. Semeval 2018 task 4: Character identification on multiparty dialogues. In *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*, pages 57–64.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation 9(8):1735-1780*, 9.

Comparison of Classification- and Regression-Based Approaches for Humor Ranking on a #HashtagWars Twitter Dataset

Luka Čupić, Vinko Kašljević, Ivan Smoković

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{luka.cupic, vinko.kasljevic, ivan.smokovic}@fer.hr

Abstract

The comparison of classification- and regression-based methods in the context of natural language processing is a rather uncommon topic of contemporary work. While there is a certain intuition behind using either of these two approaches based on a particular problem, there might be some additional information that could be obtained by using the other approach. In this work, we attempt a regression-based approach for the problem of humor ranking to compare regression and classification. We hypothesize that by training a model to predict the *measure* of humor, the model could hypothetically understand this measure and predict it, potentially providing high-quality results. By trying this approach on several models of different architectures and complexities, we conclude that although regression shows some interesting results, classification works better in most cases.

1. Introduction

Humor can be dated, at least, to thousands of years ago – the ancient Romans, for example, had a sense of humor similar to ours, showing obvious examples of irony and sarcasm (Harvey, no date). Humor is treated as a characteristic human trait that arises naturally (for most) and is generally considered straightforward (excluding some extreme variants of sarcasm and such). In contrast, computers cannot understand humor the way humans do for obvious reasons.

In this work, we study the effects of using a regression-based approach for the problem of ranking humor on a collection of tweets. The tweets we use represent users' replies to hashtags from Comedy Central's @midnight show, which features a recurring #HashtagWars game. The game host presents a certain hashtag to which the contestants reply with witty responses. For example, some real responses for the #PrisonBooks hashtag are *100 Years of Solitary*, *Hijacker's Guide to the Galaxy*, and *Slaughterhouse Five... Three With Good Behavior*. Arguably, this kind of humor may be quite difficult for a model to learn, making the problem more interesting.

In this paper, we investigate the similarities and differences between classification and regression in terms of the model's output. An intuitive (and natural) approach (as has been done in most previous work) would be to represent the humor ranking problem in terms of classification. However, because of the nature of the used #HashtagWars dataset (described in detail in Section 3.), we hypothesize that such a task might be more appropriate for a regression-based approach. We also hypothesize that doing so could enable the model, instead of simply learning the correct class labels (as is the case with classification), to be able to predict the *humor measure* for each of the tweets. Working with a continuous scale rather than with discrete labels would introduce inter-class measures, which could potentially improve the model's accuracy. Additionally, in the context of our approach, we are actually solving a proxy problem: by learning the real-valued similarities between existing tweets, we hope to be able to apply these results to solving our original problem of ranking tweets. In the rest of this work, we present our approach for undertaking the described challenge of humor ranking.

2. Related Work

There have been different approaches to solving the humor ranking problem from the SemEval's competition. For instance, one of the competing teams, (Baziotis et al., 2017), used a Siamese architecture with bidirectional Long Short-Term Memory (LSTM) networks (Bromley et al., 1994); this approach won them 2^{nd} place on the competition, while their later improvement achieved state-of-the-art results on the #HashtagWars dataset. Another approach, which was the winning model at the competition, is an ensemble of a character-based convolutional neural network and an XGBoost (gradient boosting) model (Donahue et al., 2017).

Apart from the SemEval's competition, there have also been notable attempts at humor detection and ranking. Prior to SemEval's competition, most work on humor detection focused on detecting whether something *was* or *wasn't* humor. In other words, the majority of attempts were focused on building a binary classifier. Post competition, the contestants (as well as many others) have been inspired to perform humor ranking on a larger scale, thereby producing more information about the relationship between pairs of humor units (in this example, humorous tweets). However, a great many of them had approached the humor ranking as a classification problem, which is rather reasonable (Ortega-Bueno et al., 2018; Cattle and Ma, 2018; Donahue et al., 2017; Baziotis et al., 2017). Our interest, however, was to see how this could be posed as a regression problem, and whether the results would potentially outperform the classification-based approaches.

3. Dataset

The dataset used in this paper is provided as-is on the SemEval 2017 competition website and consists of training,

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 18–21, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

evaluation, and trial data.[1] We use these sets for training, testing, and validation respectively. The dataset contains 112 Twitter hashtags in total, with each of them having, on average, 114 associated tweets. Our training, testing, and validation sets consist of 101, 6, and 5 hashtags, respectively. In terms of the number of tweets, there are 11,321 tweets in our training set, 749 in our test set, and 660 in our validation set.

Each of the tweets represents a witty response to a specific hashtag given by a viewer of the #HashtagWars game. The tweets are annotated as follows: *0* means that the tweet *did not* reach the top 10; *1* means that the tweet *is* among the top ten tweets (but not the best one), while *2* means that the tweet has been chosen as the best one, i.e. the winning tweet for a given hashtag. The inherent natural ordering of these instance labels (*0*, *1*, and *2*) was the main reasoning behind our approach: instead of simply predicting the classes, the regression-based model might be able to recognize this ordering and potentially provide us with something more useful, such as a humor metric.

3.1. Pre-processing

For classification-based approaches, tweets were first grouped into pairs containing the texts of both tweets. Each pair consists of two tweets from the same hashtag, but of different humor ranking. For instance, a tweet that belongs in the "top 10" ranking could be paired with a tweet that belongs to the "below 10" ranking, or with the "top 1" tweet. After pairing, each tweet pair is flipped with a probability of 0.5 and a label indicating which of the tweets has the higher ranking is added. The reasoning behind the flipping is to avoid having pairs that always have the higher ranked tweet on one side, which would be trivial for any model to learn and overfit to. The value of the label is "0" if the first tweet is ranked higher, or "1" if the second tweet in the pair is ranked higher. Our classification-based models take the pre-processed tweet pairs and try to predict the ranking labels. There are 103124, 6944, and 6145 pairs of tweets in the pre-processed training, testing, and validation sets, respectively.

For regression-based approaches, each tweet from the dataset was assigned a label denoting the ranking for its hashtag. A tweet ranked "top 1" would be assigned a label of value "2", a tweet ranked "top 10" would be assigned a label of value "1" and a tweet ranked "below 10" would be assigned a label of value "0". The classes are then balanced by duplicating tweets of classes that have fewer examples than the most numerous class. This is repeated until all classes have the same number of examples. After this, the labels of all tweets are normalized to a range between 0 and 1. Our regression-based models take the tweet text as input and try to predict the normalized label, which is a proxy value for the comedic value of the tweet. For regression, our training set consists of 30966 tweets, whereas the validation and the test sets are identical to those from the classification approach.

[1] http://alt.qcri.org/semeval2017/task6/index.php?id=data-and-tools

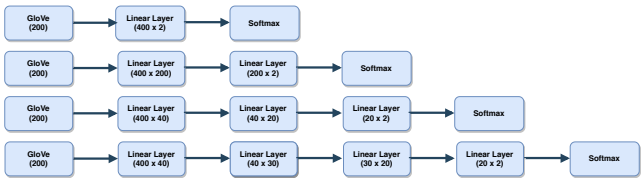


Figure 1: Architectures of classification-based models with GloVe embeddings. All layers imply a ReLU activation function.

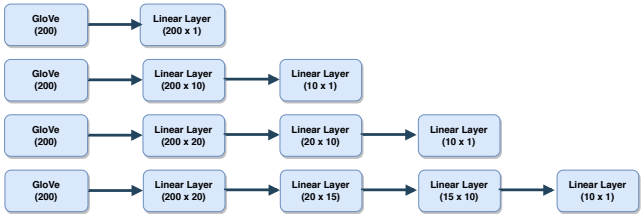


Figure 2: Architectures of regression-based models with GloVe embeddings. All layers (including the output) imply a ReLU activation function.

4. Approach

With our approach, we have tried a variety of available models. To start with, we defined the lowest accuracy benchmark (50%) with a random baseline model. We used several fully-connected models with GloVe Twitter embeddings (hereafter: GloVe embeddings). Furthermore, we used LSTM with both character and GloVe embeddings. We chose these models because they offer different layers of complexity and we wanted to see how model complexity affects accuracy for both regression and classification. In the rest of this section, we will describe these models in detail.

4.1. GloVe Embeddings

GloVe (Global Vectors) represents an unsupervised model for transforming words into vector-space (Pennington et al., 2014). We use GloVe (specifically, pre-trained word vectors for Twitter) as our main (most commonly used) feature extractor.[2]

In the context of GloVe embeddings, we use several models of increasingly higher complexities. For both classification and regression, we use GloVe embeddings of size 200 and a dropout value of 0.5. For each of the approaches, we constructed four fully-connected neural networks whose architectures can be seen in Figures 1 and 2. Figure 1 shows the models used for classification, while Figure 2 shows the models used for regression.

4.2. ELMo Embeddings

In addition to the GloVe embeddings, we also use ELMo. We use fully-connected models with ReLU activations and a dropout of 0.5. The layers' architectures (i.e. outputs and inputs) are similar to the architectures shown in Figures 1 and 2 but are left out for the sake of brevity.

[2] https://nlp.stanford.edu/projects/glove/

Table 1: Comparison of classification and regression outputs for the used models and word embeddings. The first column represents the models used in this work; *GloVe* and *ELMo* refer to the GloVe and ELMo embeddings used on the models' inputs, whether on tokens or lemmas, and the final part represents the number of fully-connected hidden layers.

Model	**Classification**			**Regression**		
	ACC (Val)	ACC (Test)	NLL	ACC (Val)	ACC (Test)	MSE
Random Baseline	50.00	50.00	-	50.00	50.00	-
GloVe Token 1FF	62.64	50.74	0.6472	53.57	58.38	0.2268
GloVe Token 2FF	**64.97**	57.40	0.5051	56.35	55.45	0.1080
GloVe Token 3FF	64.02	52.14	**0.4953**	62.41	48.30	0.0849
GloVe Token 4FF	63.62	53.01	0.4994	**67.03**	53.49	**0.0640**
ELMo Lemma 1FF	61.87	62.08	0.6098	57.47	49.03	0.1865
ELMo Lemma 2FF	61.64	62.78	0.5977	56.37	56.43	0.1405
ELMo Lemma 3FF	62.06	62.42	0.6038	57.75	49.15	0.1178
Char LSTM	58.14	**68.21**	0.5908	55.87	**61.79**	0.1802
GloVe LSTM	64.17	61.18	0.5559	60.57	50.84	0.1245

4.3. Sequential Models

Finally, we also use two recurrent networks: character- and GloVe-based LSTM. The LSTM is used to produce embeddings, which we use as input to two fully-connected layers with ReLU as activation function and a single layer of dropout at the beginning. We get the embeddings by only using the last output of the LSTM. We used this model for both classification and regression; the only difference is that the number of weights for classification is twice as big as for regression. Additionally, classification also has the softmax function on the output.

4.4. Additional Approaches

Neutral Dataset Alongside the dataset described in the previous subsection, we have also included a dataset from another SemEval competition, designed for the task of performing sentiment analysis for a given collection of tweets.[3] This dataset contains annotated tweets distributed among three sentiment categories: positive, neutral, and negative. Since our main dataset and this one are contextually similar (in terms that they are both comprised of tweets), we added this dataset as a new category, which would represent neutral tweets. Our presumption with this decision was that the model could be trained to learn the "ground truth" for neutral tweets, so that it does not become particularly overfit to humorous tweets, thus enabling the model to become more robust for future predictions.

5. Experiments and Results

In this section, we describe the learning process for our models and present and discuss the obtained results.

As mentioned in Section 3., we use the training set (consisting of 11321 tweets distributed among 101 different hashtags) for training our models. We use batching, with each training instance using a batch-size of 5,000 samples. We also use the Adam optimizer with a learning rate of 0.001 and a weight decay of 0.001. The dropout value, as mentioned earlier, is fixed at 0.5. As for the initial parameters, all of them are randomly generated using a random number generator with a fixed seed value (100) to achieve reproducibility.

We use accuracy as the evaluation metric for both classification and regression approaches. For regression, as previously mentioned, we use training and validation sets identical to those from the classification approach. The accuracy is computed by first predicting the humor score and then using the argmax function to determine which of the tweets is more humorous, giving a label to the pair, as described in 3.1.

Table 1 shows our main results (in terms of the accuracy evaluation metric), which indicate that classification outperforms regression in the majority of cases. Further looking at the results, some more interesting figures can be seen from the table. Regression-based GloVe Token 3FF, for example, produces a 62.42% accuracy on the validation set, and a mere 48.30% on the test set. Another interesting example is with classification-based ELMo Lemma 1FF model, producing 62.08% on test set, unlike regression, which performs worse than the Random Baseline. The results are somewhat better for the Char LSTM model, but there is still a 6.42% difference between classification (68.21%) and regression (61.79%).

Unable to provide any tangible intuition for the achieved results, we provide some possible explanations: (i) regression-based approaches use considerably less data than classification-based ones; (ii) dimensionality of the input data for regression is half that of the dimensionality for classification; (iii) regression is trained on a different (proxy) problem compared to classification; and (iv) the dataset used for regression is somewhat perplexing. The last point needs some further elaboration: if the model is trying to learn a certain distribution on the train set, choosing a model from the second (test) distribution, and then measuring the accuracy on the third (validation) distribution, the results might not be satisfactory enough. In contrast to this, classification-based approaches might be able to overcome this problem due to having two tweets at the

[3] https://www.dropbox.com/s/byzr8yoda6bua1b

input.

We hypothesize that the problem might lie in the dataset. Looking at the results from Table 1, it is evident that in some cases, there is an inconsistency between accuracy on validation and test set; this was certainly an unwelcome surprise. GloVe Token regression model, for example, achieves poor accuracy on the validation set, but good accuracy on the test set.

Finally, when we take into account all previously mentioned issues with regression, it is not surprising that classification works better. Still, both regression and classification achieve some interesting figures.

5.1. Additional Results

Our additional approaches have, unfortunately, provided unsatisfactory results. The neutral dataset has been shown not to work as we expected, so we disregarded it in the early stages of our work.

6. Conclusion

In this work, we compared classification- and regression-based approaches in the context of humor ranking. We used a variety of models and word embeddings on the SemEval's dataset of humorous tweets. With the achieved results, we showed that while classification generally works better than regression for humor ranking, additional research could be performed on the subject of comparing regression and classification because we believe that there might be a particular subset of NLP problems in which the presented ideas and hypothesizes could be more applicable.

Acknowledgements

We would like to take this opportunity to express our endless gratitude to our dear colleague Ivan Smoković for leaving us to work in peace and quiet while he went to a rock concert in Vienna.

References

Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. DataStories at SemEval-2017 Task 6: Siamese LSTM with Attention for Humorous Text Comparison.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Sickinger, and Roopak Shah. 1994. Signature Verification using a "Siamese" Time Delay Neural Network.

Andrew Cattle and Xiaojuan Ma. 2018. Recognizing Humour using Word Associations and Humour Anchor Extraction.

David Donahue, Alexey Romanov, and Anna Rumshisky. 2017. HumorHawk at SemEval-2017 Task 6: Mixing Meaning and Sound for Humor Recognition.

Brian Harvey. no date. Graffiti from Pompeii. http://www.pompeiana.org/Resources/Ancient/Graffiti%20from%20Pompeii.htm.

Reynier Ortega-Bueno, Carlos E. Muñiz-Cuza, José E. Medina Pagola, and Paolo Rosso. 2018. UO UPV: Deep Linguistic Humor Detection in Spanish Social Media.

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.

Automated Detection of Hyperpartisan News Articles

Ilija Domislović, Tonko Sabolčec, Robert Tušek

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ilija.domislovic, tonko.sabolcec, robert.tusek}@fer.hr

Abstract

This document presents a system design of two models that are used for hyperpartisan news article detection and classification of articles as left-oriented, center or right-oriented. The first model is based on TF-IDF word weighting and fastText document representation and the second model is inspired by deep learning architectures of CNN and BiLSTM. The models are trained and tested on SemEval competition 2019 task 4's dataset and results indicate that the first model is quite accurate when compared to state-of-the-art methods.

1. Introduction

In the context of journalism, partisanship refers to the ideology of strongly supporting a particular person or political party, often without even judging the veracity of the matter. In extreme cases, authors produce hyperpartisan news articles, to manipulate society and impose their own ideology. Hyperpartisan news, along with fake news, are affecting our opinions and lives not only in a political sense but also in a non-political sense, changing our identity; people who accept the imposed ideology are usually getting more and more angry at their opponents.

With modern technology, distribution and flow of information are rapidly progressing. It's also easier to target certain people with certain information, which makes it easier for hyperpartisan news to reach targeted people and manipulate them. An example from recent history is the case of the US presidential elections in 2016, which created several studies and theories about factors related to Trump's victory, e.g. Faris et al. (2017) showed that the majority of media was negative for both candidates, but largely followed Trump's agenda. At the same time, an increasing amount of news makes it difficult to manually analyze and verify the information. To solve the problem, a growth in interest to design an automated system for detecting hyperpartisan and fake news articles has emerged in recent years.

In this work, we base our research on the SemEval 2019 competition task 4 by Kiesel et al. (2019). We present two supervised machine learning (ML) architectures which are trained to solve two tasks. The first task is to classify a given article as either hyperpartisan or not hyperpartisan. The second task is to assign the given article political orientation: left-biased, center-left, center, center-right and right-biased. Models are trained on the SemEval 2019 competition task 4's dataset for hyperpartisan news detection and are as accurate as the ones submitted by participants of the competition. Our goal in this work is to find out whether we can determine the hyperpartisanship of publishers based on the articles they publish.

2. Related Work

Detection of hyperpartisan news articles is a text classification problem at its core. In order to make use of machine learning methods, the first step is to transform documents into a representation suitable for ML algorithms, i.e. to extract features from documents. Traditional approach is to use bag-of-words representation of text, i.e. frequency histogram of most common words in the dataset. Often, this method is used alongside bag-of-n-grams, where n-gram is a sequence of n consecutive words, characters, etc., which are able to capture shorter patterns. On top of that, the Term Frequency – Inverse Document Frequency (TF-IDF) method, which was used by Ramos (2003) to determine word relevance in documents, can be used to improve results even further.

Another way to represent a text is by representing each word as low-dimensional vector (typically hundreds of elements), so-called word embeddings, and then summarizing all document's word vectors. Word embeddings were introduced by Mikolov et al. (2013), and are nowadays present in many natural language processing (NLP) libraries, e.g. Gensim's word2vec by Rehurek and Sojka (2010). Other representations include Facebook's fastText by Bojanowski et al. (2016) and GloVe by Pennington et al. (2014).

An increase in popularity of neural networks in recent years encouraged its use in various NLP tasks, including text classification. Two most common architectures are based on convolutional neural networks (CNN) by LeCun et al. (1989) and recurrent neural networks (RNN) by Jordan (1990). CNNs proved to be good for pattern matching, e.g. extracting interesting n-grams from the text, while RNNs are used for processing sequences and are capable of keeping track of arbitrary long-term dependencies in the input sequence. Long short-term memory (LSTM) Hochreiter and Schmidhuber (1997) models are often seen as an improvement of vanilla RNN and used as the default recurrent method. Zhou et al. (2015) combined the two deep learning architectures and introduced convolutional LSTM (C-LSTM).

When it comes to verifying the truthfulness of text, there are three main approaches that are commonly used: knowledge-based, context-based and style-based deception detection as stated in Potthast et al. (2018). Knowledge-based approach retrieves various information from the knowledge base, e.g. the Internet and uses it to verify claims in a specific news article. However, this method does not work well for new claims and requires more time to be implemented compared to other methods. The

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 22–25, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

context-based approach uses meta information such as the author's or newspaper company's information. For example, Mocanu et al. (2015) showed that users more prone to interact with false claims are usually exposed to conspiracy groups on Facebook. Even though it results in quite accurate predictions, this method requires a lot of training data and lacks the integration of article's content itself. Finally, the style-based approach assumes that real-life and self-experienced events differ from imagined events in terms of content and quality, i.e. that deception detection can be achieved by simply analyzing the style and structure of the text. As an example, Potthast et al. (2018) reported on comparative style analysis of hyperpartisan news and fake news. Some of the features they used in their work include most common 1-grams, 2-grams, 3-grams of characters, words (excluding stopwords) and part-of-speech (POS) tags, length of news article, number of paragraphs and hyperlinks in the article and common style metrics such as Readability Index, Coleman Liau Index, etc.

To explore hyperpartisan news article analysis even further, this problem was given as the 4th task on SemEval 2019 competition, in which over 40 teams participated.

3. Models

In this section, we will introduce two ML architectures used for this task. The first one is based on simple feature extraction and logistic regression, while the other uses deep learning paradigm.

3.1. Shallow Model

Following work of traditional methods, we designed a simple model based on TF-IDF and fastText representation of the document. Features of both methods are concatenated and propagated to logistic regression for classification.

TF-IDF is an information retrieval method that is used to weight words in documents. It consists of two components: term frequency (TF) and inverse document frequency (IDF). Term frequency is a metric that represents number of appearances of a certain term in the document. Inverse document frequency measures how important that term is in the whole corpus, e.g. stopwords such as *the* are irrelevant because they almost certainly appear in every document. Most popular schemes for calculating TF and IDF are:

$$\text{TF}(k_i, d_j) = 0.5 + \frac{0.5 \cdot freq(k_i, d_j)}{\max_{k \in d_j} freq(k, d_j)}$$

$$\text{IDF}(k_i, D) = \log \frac{|D|}{|\{d_j \in D | k_i \in d_j\}|},$$

where D represents a set of all documents in the corpus, d_j a single document, and k_i a single term. Weight for word k_i in the document d_j is then calculated simply as the product of two components: $w_{i,j} = \text{TF}(k_i, d_j) \cdot \text{IDF}(k_i, D)$.

We used TF-IDF weighting scheme over X most frequent 1-grams, 2-grams, 3-grams and 4-grams of words in the training corpus.

FastText by Bojanowski et al. (2016) is a method for representing a word as a vector, acknowledging morphology of the word. We convert document to vector by averaging fastText vectors of all the document's words.

Based on title and content of the articles, we extract four sets of features: title's TF-IDF vector, content's TF-IDF vector, title's fastText representation and content's fastText representation. Final feature set is obtained by concatenating the four mentioned ones. To classify features into one of 2 or 5 final categories (depending on the task) we use simple logistic regression with L1 regularization.

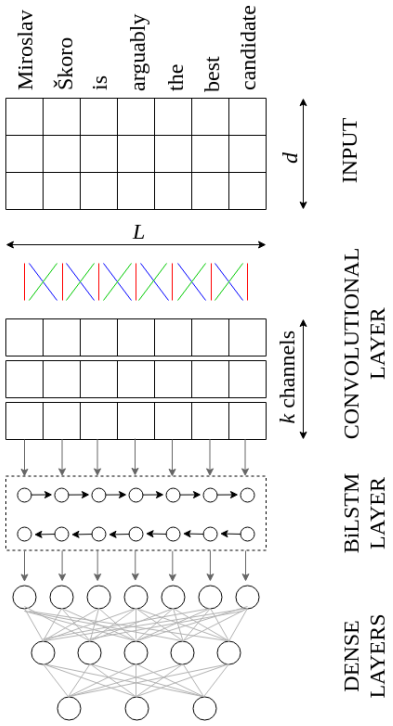


Figure 1: Convolutional LSTM architecture.

3.2. C-LSTM Model

Inspired by work of Zhou et al. (2015), we decided to build a C-LSTM model for hyperpartisan text classification. The architecture of this deep neural network is illustrated in the Figure 1. Two main components of this network are CNN which captures relevant patterns in the input text and bidirectional LSTM (BiLSTM) which can model features based on context of patterns that were extracted from CNN.

The input to the neural network is a fixed size sequence of words. We denote that length as L. For word representation in the sequence we use word embeddings, i.e. each word is represented as a vector of size d. Since news articles are not of a fixed size, we use first L words of the news article, excluding the stopwords. In some cases, articles are not long enough to fit the length of L. Here, we use padding to fill remaining parts of the input sequence with null-vectors.

In our implementation the length of input sequence was set to $L = 300$ words and we used GloVe pretrained word vectors of size $d = 50$.

Convolutional layer is used to capture interesting patterns from the sequence. As suggested by the original work

Zhou et al. (2015) we use only one convolutional layer without any pooling method as that could break sequence organization due to the discontinuity of selected features. We used 32 filters and kernel of size 5, and ReLU as the activation function.

Output of the convolutional layer is passed to BiLSTM layer. Recurrent neural models turned out to be suitable for sequence-based inputs, such as text, because of ability to keep track of context history. Here, we used bidirectional LSTM layer with hidden state of size 100.

Finally, two dense layers follow. The first dense layer extracts features from the output of BiLSTM layer and the second dense layer combines them to form one of 2 or 5 classes (depending on the task). Size of the first layer was set to 32 and we used ReLU as activation function. Second layer uses *softmax* activation function. For regularization, we used dropout from both dense layer with the rate of 0.3.

4. Dataset

To train and test our models we used the data provided by competition organizers. The data is provided in a XML format and is split into two datasets.

The first dataset contains 645 articles. Each article is labeled as hyperpartisan or non hyperpartisan. The labeling of the articles was done through crowdsourcing. Out of all the articles, 238 are labeled as hyperpartisan and 407 are labeled as non hyperpartisan.

The second dataset contains 750 000 articles. Each article is labeled with one of five biases: left, left-center, least, right-center and right. Half of the dataset is hyperpartisan and half is not. The left and right biases are considered hyperpartisan and each has 187 500 articles in the dataset. The labeling of this dataset was done by using the MediaBiasFactCheck.com website and with the help of BuzzFeed journalists. Articles in this dataset are labeled automatically based on the bias of their publisher. This can be a problem since articles with no political connotations will be labeled as hyperpartisan because their publisher is hyperpartisan.

5. Evaluation

For evaluation purposes, we created four different models. For each dataset we created a shallow and a C-LSTM model. The small dataset only states if the article is hyperpartisan or not, therefore models trained on this dataset perform binary classification. The large dataset has more fine-grained labels that show the intensity as well as orientation of the partisanship. Binary classification can also be done on this dataset, but because of the way the data was labeled we decided to do multi-class classification on this dataset.

Finally, we had to clean the data because the organizers decided to preserve the metadata from the HTML DOM of the articles. This means that we removed HTML tags as well as HTML escape characters from the dataset.

The large dataset was not used in its entirety because we lacked the required computational power. For the shallow model we created five smaller datasets where each dataset contains 50 000 articles. An article was not present in more than one dataset. Each dataset was split in an 80:20 train-test ratio. Because of the time complexity of our deep model we trained on even smaller dataset, containing 1000 articles split in the same train-test ratio.

6. Results

6.1. Binary Classification

The binary classification task was performed on the small dataset. The shallow model produced an accuracy score of 0.764 on this task which is comparable to the accuracy of the top 10 models in the SemEval competition. Our deep model trained on the small dataset produced results with lower accuracy than our shallow model.

Table 1 shows comparison of our models with models submitted on the SemEval 2019 competition [1] on the small dataset. However, it should be noted that results are not tested on the same dataset, as we did not have access to the official test set and used subset of the train set for testing purposes.

Table 1: Accuracy of our models compared to top 5 models submitted on SemEval 2019 competition for binary classification.

Model	Accuracy
Jiang, Petrak, et al.	0.822
Srivastava, Hyang Kim, et al.	0.820
Sasaki, Hanawa, et al.	0.809
Yeh, Tintarev, et al.	0.806
Isbister	0.803
our shallow model	**0.764**
our C-LSTM model	**0.73**

6.2. Fine-Grained Classification

The fine-grained classification was performed on the large dataset. The shallow model produces results on this dataset with an accuracy of 0.85. This model outperforms the winner of the competition for this task. The result can be seen in Table 2. We assume the reason for this is the fact that the test dataset of the competition was not available to us, so our model could not be tested on the same data.

Table 2: Accuracy of our models compared to top 5 models submitted on SemEval 2019 competition for fine-grained classification.

Model	Accuracy
Bestgen	0.706
Moreno, Hubert, et al.	0.680
Papadopoulou, Zampoglou, et al.	0.664
Zehe, Hotho, et al.	0.663
Lee, Liu, et. al	0.663
our shallow model	**0.858**
our C-LSTM model	**0.595**

[1] Full leaderboard can be found at *https://pan.webis.de/semeval19/semeval19-web/leaderboard.html*

The confusion matrix for test set evaluation is shown in Table 3. From the confusion matrix we can conclude that left-biased articles result in most of the false positives, while the center is the most separable category (with accuracy of approximately 95%). Because of the time complexity of our deep model, we trained on a smaller amount of data than the shallow model and were able to obtain an accuracy of 0.6 on the test set.

Table 3: Confusion matrix on evaluated test dataset of our shallow model where L is left bias, L-C is left center bias, C is least bias, R-C is right center bias and R is right bias.

	L	L-C	C	R-C	R
L	2210	121	65	24	136
L-C	234	823	47	30	40
C	135	30	2898	16	16
R-C	101	56	31	491	27
R	239	41	26	4	2159

6.3. Hyperparameters and Features

Using t-tests with alpha 0.05 we have shown that the shallow model with all features performs significantly better than a model where any of the features are excluded. The tests were performed using 10-fold cross-validation.

We also performed grid search on out TF-IDF vectorizer and our Logistic classifier. The best results were obtained when we used unigrams, bigrams, trigrams and 4-grams in our vectorizer and L1 regularization with the inverse regularization strength of 5 in out classifier.

7. Conclusion

In this paper, we develop two different models to tackle the problem of hyperpartisan news articles. One of the models is a shallow model that represents the traditional approach to language processing that uses TF-IDF and fastText for document representation. The other model represents the modern approach that employs deep neural networks for text classification. We show that even though state-of-the-art models use deep neural networks our shallow model outperforms our deep model. For the problem of binary classification of hyperpartisan articles, both of our models perform worse than the top performer of the competition. Our shallow model outperforms the best model of the competition in the detection of hyperpartisanship in publishers.

In the future we would like to acquire the competitions testing datasets so that we can do a proper comparison of our results with the results of the competition.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information.

Rob Faris, Hal Roberts, Bruce Etling, Nikki Bourassa, Ethan Zuckerman, and Yochai Benkler. 2017. Partisanship, propaganda, and disinformation: Online media and the 2016 u.s. presidential election.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory.

Michael I. Jordan. 1990. Attractor dynamics and parallelism in a connectionist sequential machine.

Johannes Kiesel, Maria Mestre, Rishabh Shukla, Emmanuel Vincent, Payam Adineh, David Corney, Benno Stein, and Martin Potthast. 2019. Semeval-2019 task 4: Hyperpartisan news detection.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality.

Delia Mocanu, Luca Rossi, Qian Zhang, Marton Karsai, and Walter Quattrociocochi. 2015. Collective attention in the age of (mis)information.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation.

Martin Potthast, Johannes Kiesel, Kevin Reinartz, Janek Bevendorff, and Benno Stein. 2018. A stylometric inquiry into hyperpartisan and fake news.

Juan Ramos. 2003. Using tf-idf to determine word relevance in document queries.

Radim Rehurek and Petr Sojka. 2010.

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

Stock Market Prediction from News Sentiment: Is It Possible?

Ivan Ilić, Dorian Ivanković, Davor Vukadin

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ivan.ilic2, dorian.ivankovic, davor.vukadin}@fer.hr

Abstract

Since its creation, stock market prediction has been a subject of great interest to people. With advances in the field of artificial intelligence over the last few years, and large amounts of data at our disposal, it becomes possible to experiment with many different ideas. In this paper we examine the possibility of predicting stock market movement using sentiment analysis of news. We tackle this task by using two different approaches. First we perform sentiment analysis on world daily news and examine whether it is possible to predict the movement of the DJIA stock market index using such information. In the second approach we built a domain-specific model that performs sentiment analysis on financial news related to a specific company, for which we then try to predict stock price movement.

1. Introduction

Predicting exchange rates of stock prices is an important financial task that has received an increased amount of attention in recent years. Presumably, such task is extremely difficult and challenging to tackle, mainly due to highly complex nature of financial economics, but also some other factors such as politics and natural disasters.

A lot of research has been done on this subject, mainly to see whether stock market movement can be predicted at all. On one hand, there is research that suggests that stock market cannot be predicted with accuracy of more that 50%. Such research is based on random walk theory and Efficient Markets Hypothesis (EMH). EMH (Malkiel and Fama, 1970) states that stock prices reflect all relevant information and that market cannot be predicted. Changes in the price are only due to new information and news. Since news happen randomly, stock market should follow a random walk trajectory. On the other hand, there is research that suggests markets do not follow a random walk pattern and predictions can be made to some extent (Vu et al., 2012).

In this paper we examine the possibility of predicting stock market movements from news sentiment, which ranges from negative to positive. In general, textual data is more challenging to handle than numeric data. News textual data is unstructured by nature, but it contains collective expressions that can be of great value when making financial decisions. There are two core ideas we follow. First idea we pursue is predicting stock market movement based on sentiment of daily world news. Since world news cover a lot of different categories and areas of life, we build a model that performs general sentiment analysis of news. We then use those sentiment predictions as features for modeling stock price movement. For this purpose we built a custom dataset by crawling Reddit[1] subforums and collecting the most popular world news. Next, we investigate predicting stock price movement for one particular company, based on sentiment of news regarding that company. In this case, news is usually business or finance related. Since domain of business and finance uses unique linguistic and semantic features, it is clear that sentiment analysis model trained on broad and diverse dataset will not provide us with optimal features. To overcome that, we built a domain-specific neural network model that performs sentiment analysis on business and financial news. We then apply the same idea and use those sentiment predictions as features for modeling stock price movement. We use financial news data from SemEval 2017 competition, Task5, Subtask 2 (Cortis et al., 2017) for this task.

[1] https://www.reddit.com/

2. Related work

Sentiment analysis in this task requires both domain-specific and commonsense knowledge. Previous approaches to sentiment analysis resort to domain specific, rich hand-crafted features (Abbasi et al., 2008). More recent work incorporating word embeddings (Pennington et al., 2014) eliminates the need for manual creation of domain-specific features, since this is a very time-consuming and complex task, while still obtaining very good performance. Therefore, we also utilize word embeddings in our approach.

The most intriguing application of predicting sentiment on financial news is predicting market dynamics (Goonatilake and Herath, 2007; Van de Kauter et al., 2015).

(Goonatilake and Herath, 2007) statistically analyze the influence of news on DJIA[2], NASDAQ[3] and S&P 500[4], and they were able to conclude that there is an association between news items and market fluctuations. Positive news has the capability to increase optimism among people and therefore boost the market (Van de Kauter et al., 2015). Some previous work (Bollen et al., 2011) also shows the correlation between the mood on Twitter feeds and the value of the Dow Jones Industrial Average (DJIA). This ideas lead us to our first approach where we examine the effect of world daily news on stock market movements.

It is not really clear which news affect the stock market

[2] https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average

[3] https://en.wikipedia.org/wiki/NASDAQ

[4] https://en.wikipedia.org/wiki/S%26P_500_Index

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 26–30, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

Table 1: Examples from world news training dataset

title	label
Fatal car crash kills 13 because the driver was texting	0
Canadian lottery winner donates $40 million jackpot to charity.	1

the most. (Boudoukh et al., 2013) provide a detailed analysis of possible solutions.

(Van de Kauter et al., 2015) propose a fine-grained method that detects explicit and implicit sentiment that can be used for topic-specific sentiment analysis in financial news text, using a corpus of company-specific news articles, similar to the SemEval 2017, Task 5, Subtask 2 dataset we used in our approach.

Additionally, it has been shown (Heston and Sinha, 2017) that the time-wise relationship between news sentiment and stock market movement is quite complex. The authors argue that daily news could predict stock returns for only one to two days, and that weekly news have information to predict stock returns for one quarter. In the paper it is also shown that positive news have a quick impact, and negative ones have a long-delayed reaction. We also try to take this into consideration in our approach.

3. Approach

In section 3.1. we describe in more detail the datasets we used, preprocessing methods, news sentence representation and model architectures. In section 3.2. we describe the datasets and the methods we used to try to predict stock market movements in each of the approaches mentioned.

3.1. News sentiment

Dataset. The dataset used for training the sentiment model in the first approach (daily world news sentiment) consists of monthly thread titles from two different reddit channels. In the absence of labeled data we used thread titles from reddit channels r/upliftingnews and r/morbidreality in order to get "automatic" sentiment labels and avoid having to label them by hand. Labels 1 (positive) and 0 (negative) were used for the titles from each reddit channel respectively. The complete training dataset consists of 15 000 examples (250 for each month) of titles with a positive sentiment and 15 000 examples of titles with a negative sentiment (total time crawled was 5 years: from about 2014/5 to 2018/5). 6000 out of total 30 000 examples (3000 from each class) were used for testing, while the rest 24 000 was used for training. The average length of the title is 20 words. An example of a title with positive sentiment and an example of a title with negative sentiment from the dataset are given in Table 1.

The dataset used in the second approach (financial news sentiment) is the dataset from Semeval 2017, Task 5, Subtask 2 (Cortis et al., 2017), which consists of financial news headlines crawled from several online sources such as Yahoo Finance. Each example consist of a headline, company name and a fine-grained sentiment score. An example from the dataset is given in Table 2.

Table 2: Example number 228 from the dataset

id	228
company	BP
title	BP signs $12 billion energy deal in Egypt
sentiment	0.48

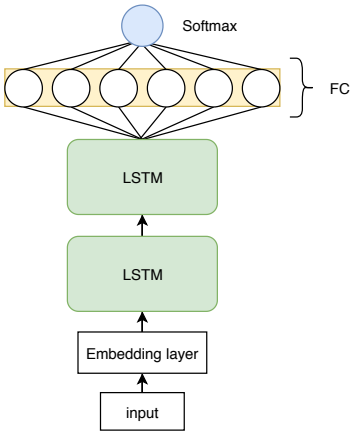


Figure 1: Architecture of the sentiment analysis model.

The sentiment score values range from -1 (negative, bearish) to 1 (positive, bullish), but we reformulate this problem as a binary classification problem by labeling sentiment scores ≥ 0 as 1 (bullish) and < 0 as 0 (bearish), since we find that 0.01 precision sentiment scores were not beneficial for our approach.

The dataset consists of 1142 news headlines with 80% used for training, and the remaining 20% used for testing. The average length of the news headline is about 10 words, and there are 294 unique companies in the dataset. The number of bullish headlines is 691 and the number of bearish headlines is 451.

Preprocessing. It is quite intuitive that named entities do not have a major impact on domain specific news sentiment. Therefore we utilize NLTK's(Loper and Bird, 2002) named entity recognition to filter out named entities from the titles of the Semeval dataset. All numbers were also replaced by "number" to reduce noise.

Additionally, in the Semeval dataset the name of the company associated with each title is also provided; we use a simple regular expression matching method to remove this information from the title, since named entity recognition does not always perform this part correctly. This way we make the approach more general, since this information is not relevant for the task at hand.

The following was utilized in preprocessing of both datasets: sentence tokenization using spaces as separators, lower case conversion, lemmatization (using NLTK) and stopword and punctuation removal.

News representation. The words of the title were represented as fixed length vectors using pretrained word embeddings - Glove (Pennington et al., 2014). Word embeddings are task independent features, and they are often used in many NLP tasks.

Architecture. The sentiment model architecture is

Table 3: Stock market values example

Open	High	Low	Close	Volume	Adj Close
17924.24	18002.38	17916.91	17949.37	82160000	17949.37

shown in figure 1. The model is a two layer LSTM (Hochreiter and Schmidhuber, 1997) followed by a dense layer and a softmax output. The model used for sentiment of daily world news uses an embedding layer of size 100 and does not use the fully connected layer, only the two LSTM layers of dimension 64, followed by an softmax output unit. The model used in modeling domain specific news sentiments uses an input embedding of size 300 and the extra fully connected layer of size 32 with ReLU activation on top of two LSTM layers of size 32.

Before moving on to sentiment-stock movement modeling, the respective model is trained on the whole respective dataset to make use of the complete dataset.

3.2. Stock market prediction

Relying on the already described component for sentiment analysis, we moved on to investigate several approaches for predicting stock market movement using transfer learning.

As mentioned earlier, time-wise relationship between news sentiment and stock market movement is quite complex. We treat this as another hyper-parameter, which we denote as T. When T equals 1, it means we only use data from the current day to model this relationship. When T is 5, for example, then we are considering the data for the current day, as well as previous 4 days.

Before moving on to more detailed descriptions of both scenarios, we give a quick description of the datasets we used. The daily world news dataset (Sun, 2016) we use for the first approach was obtained by crawling top 25 thread titles from the r/worldnews subreddit for each day, ranked by their score. The sentiment features for these titles are extracted using the previously trained model for daily world news sentiment. To get the output stock movement label for each day, we use the following rule to get a binary output: 1 if the DJIA index value rose or stayed the same for the day (close - open), and 0 if the value decreased. The complete dataset consists of 1990 examples (25 headlines for each example) for days ranging from 8.8.2008 to 1.7.2016.

An example of DJIA data for one day is shown in Table 3. We try to use only sentiment for stock market movement prediction, so fields besides open and close which are used for calculating the label (1 = increase, 0 = decrease), are not used. The dataset for single company stock movement consists of stock movement labels which are defined the same as in the dataset for the first approach, with the key difference being that the stock prices were obtained for a single company. In our experiments we chose Apple Inc. For the financial news titles, we crawled 10 thread titles for each day from the Yahoo finance Apple Inc. site for 2016-2019. The final dataset consists of 7653 titles.

3.2.1. Predicting general stock market movement

In this approach we try to predict the direction of market movement based on daily world news sentiment. For each day a collection of world news is used as input for the sentiment analysis model which classifies them as 0's (negative) or 1's (positive). We use the number of positive news for each of T days (the current day and T-1 previous days) normalized to [0, 1] interval as features for predicting the stock movement of the current day, where we experiment with different values of T. One training example is shown in Table 4, when $T = 2$. The number in the parentheses shows the number of positive news, which was then normalized to [0, 1] by diving by the total number of the news for the day, which is 25 in this dataset.

Table 4: Training example for predicting DJIA movement when $T = 2$.

# positive news from today	0.76 (19)
# positive news from prev. day	0.88 (22)
DJIA movement for today	1

Table 5: Training example for predicting stock price movement for Apple Inc. when $T = 2$.

# positive news from today	0.7 (7)
# positive news from prev. day	0.5 (5)
Apple Inc. stock price movement for today	1

3.2.2. Predicting single company stock movement

Unlike the previous approach, this approach only uses data for Apple Inc. News related to that company are fed to the domain-specific sentiment model, and we again count the number of positively classified news. This time we divide the total number of positive news by 10, since there are 10 news in the dataset for each day. The reasoning behind this approach is that by using only a single company and only news related to it, the information we get from each title sentiment is more likely to correlate with the stock movement of that particular company. One training example is shown in Table 5.

4. Evaluation and results

As mentioned before, we tackled the problem of obtaining the sentiment labels from news titles by creating two deep neural models, one for each of our sentiment analysis datasets. For the first, global news dataset model, we use Adam optimizer (Kingma and Ba, 2014) with default parameters. Both LSTM layers were regularized by using dropout with keep probability of 0.8 (Srivastava et al., 2014), while only the second layer used a L2 regularizer with a factor of 1e-5. For the dense layer, we used the sigmoid activation with no regularization. This model was trained for 15 epochs.

For the Semeval dataset model, we also used Adam optimizer with learning rate set to 0.01. The two LSTM layers were regularized using L2 regularization with 1e-5 regularization factor and dropout with keep probability of 0.5. The dense layer's weights were regularized using L2 regularization with 1e-5 regularization factor. The model was trained for 25 epochs.

For stock market prediction, we used a number of commonly used Machine Learning models: logistic regression, k-NN classifier, Naive Bayes classifier, Random forest with

Table 6: Ablation study: average accuracy for each of the models (FP=full preprocessing)

	reddit	SemEval
FP + GloVe	**0.921**	**0.806**
- GloVe	0.913	0.753
- FP	0.903	0.752
Baseline	0.813	0.704

the number of estimators set to 100, SVM and a deep neural network. The network consists of two fully connected layers of size 64 using ReLU activation and an output layer of size 1 with a sigmoid activation. L2-regularization was used on each of the layers with a factor of 0.001 and 0.8 dropout was applied for both fully connected layers. Adam optimizer was used with default parameters. We trained the DNN for 15 epochs.

Evaluation was done by performing 5-fold cross-validation for each of the models mentioned in this section. We used accuracy as an evaluation metric. Afterwards, significance testing was used to test whether the models perform significantly better than the baselines and the ablated models (t-test). All performed tests were two-sided and the used significance level was 0.05.

4.1. Sentiment analysis evaluation

As a baseline for our sentiment models we used a simple bag-of-words approach. For each title we performed tokenization and stop word removal. Then, we selected 5000 most frequent words found in the training set and used them as features in the following way: each title was represented as a 5000-dimensional vector, where each element of the vector indicates whether a word from the title is in the 5000 most frequent words. A NB classifier was trained on this data. We compare the performance of the baseline to each of our LSTM models. We found a significant difference in performance of our models compared to the baseline for each task.

Ablation study was also performed for both of our sentiment models by first removing the pretrained GloVe embeddings and then by removing the preprocessing. The global news sentiment model is significantly different from both ablated models. The domain specific sentiment model is, similarly, significantly different from both other ablated models. Interestingly, we do not report preprocessing having a significant impact on the performance in this case. We hypothesize that the reason behind this is the short length of the titles themselves and all of them having a very similar structure. Another reason behind this might be the undersized dataset in which leaving out preprocessing actually leads to overfitting, since all titles have a similar structure. We assume that preprocessing would have a greater impact with sufficient data, as was the case with the global news sentiment.

4.2. Stock prediction evaluation

We experiment with sequence lengths set to $T = 1$, $T = 5$ and $T = 10$ for our approach described in 3.2.1. and 3.2.2. The results are shown in Table 7 and Table 8. Since Logistic Regression had the best performance on both $T = 5$ and $T = 10$ sequence lengths, we chose this model in order to perform significance testing against the baselines.

Table 7: Average accuracy for DJIA

Model	$T = 1$	$T = 5$	$T = 10$
LogReg	0.532	**0.537**	**0.536**
kNNClf	0.485	0.497	0.479
NaiveBayes	**0.535**	0.531	**0.536**
RandomForest(100)	0.520	0.489	0.496
SVM	**0.535**	0.511	**0.536**
DNN	**0.535**	0.529	**0.536**

Table 8: Average accuracy for Apple Inc.

Model	$T = 1$	$T = 5$	$T = 10$
LogReg	0.529	**0.534**	**0.532**
kNNClf	0.491	0.493	0.483
NaiveBayes	0.530	0.527	0.535
RandomForest(100)	0.511	0.492	0.499
SVM	**0.532**	0.521	0.528
DNN	**0.532**	0.527	0.529

Two baselines were used in order to evaluate our stock prediction models' performance. First one was a simple uniform probability dummy classifier, which outputs the label at random following a uniform distribution. The second one was a most frequent dummy classifier, whose outputs are labels of a class that is most frequently found within the training data. We performed significance testing on the Logistic Regression model for the sequence length of $T = 10$ and compared it to both of the baselines. We cannot reject the null hypothesis at the used significance level, so the performances are not significantly different.

5. Conclusion

In this paper we attempted to predict stock market movement based on sentiment analysis of news. News we analyzed were daily world news and financial news specific to a single company. In both cases, we first built a model that performs sentiment analysis and then we move on to stock market prediction using sentiment predictions as features. It turned out that it might not be possible to accomplish such a task using daily world news nor financial news alone. We argue that using news sentiment solely is simply not enough information to model such complex interactions of many different factors that affect the stock market. More fine grained sentiment analysis might be more beneficial, for example 5-classes ranging from very negative to very positive, as well as modeling more topic specific sentiment. It would also be interesting to see how this task could benefit from other information such as financial reports and records, which we leave for future work.

References

Ahmed Abbasi, Hsinchun Chen, and Arab Salem. 2008. Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums. *ACM Transactions on Information Systems (TOIS)*, 26(3):12.

Johan Bollen, Huina Mao, and Xiaojun Zeng. 2011. Twit-

ter mood predicts the stock market. *Journal of computational science*, 2(1):1–8.

Jacob Boudoukh, Ronen Feldman, Shimon Kogan, and Matthew Richardson. 2013. Which news moves stock prices? a textual analysis. Technical report, National Bureau of Economic Research.

Keith Cortis, André Freitas, Tobias Daudert, Manuela Huerlimann, Manel Zarrouk, Siegfried Handschuh, and Brian Davis. 2017. Semeval-2017 task 5: Fine-grained sentiment analysis on financial microblogs and news. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 519–535.

Rohitha Goonatilake and Susantha Herath. 2007. The volatility of the stock market and news. *International Research Journal of Finance and Economics*, 3(11):53–65.

Steven L Heston and Nitish Ranjan Sinha. 2017. News vs. sentiment: Predicting stock returns from news stories. *Financial Analysts Journal*, 73(3):67–83.

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

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

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*.

Burton G Malkiel and Eugene F Fama. 1970. Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417.

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.

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.

J. Sun. 2016. Daily news for stock market prediction, version 1. 1, August.

Marjan Van de Kauter, Diane Breesch, and Véronique Hoste. 2015. Fine-grained analysis of explicit and implicit sentiment in financial news articles. *Expert Systems with applications*, 42(11):4999–5010.

Tien Thanh Vu, Shu Chang, Quang Thuy Ha, and Nigel Collier. 2012. An experiment in integrating sentiment features for tech stock prediction in twitter. In *Proceedings of the workshop on information extraction and entity analytics on social media data*, pages 23–38.

Repetitiveness in Lyrics: an Insight in Music Genre Classification

Josip Jukić, Jeronim Matijević, Mate Mijolović

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{josip.jukic,jeronim.matijevic,mate.mijolovic}@fer.hr

Abstract

This work proposes two different approaches to music genre classification task based on lyrics. Firstly, model grounded on *fastText* embeddings was introduced, which ultimately delivered the best performances. As a different setup regarding this matter, we offered a model with custom crafted features, constructed to capture repetition structures in lyrics. Consensus clustering algorithm was considered as a feature selector in this process. Importance of repetitiveness in lyrics is further explored, especially in the view of genre classification. We shaped this task to utilize convolutional neural networks in lyrical pattern recognition. Performances of these models were evaluated on two different datasets: MetroLyrics and MusixMatch which contain 10 and 4 different genre types respectively. Our work tends to indicate that lyrical repetition can be exploited in genre classification, especially in *hip-hop*, *pop* and *R&B*.

1. Introduction

Managing your music libraries can be a tedious task. This is posing an increasingly hard problem nowadays because of the growth of online music databases and easy access to music content. One way to categorize and organize songs is based on the genre. Although the division of music into genres is somewhat arbitrary and subjective, a particular genre can still be described within some characteristics of the music such as rhythmic structure, harmonic content and instrumentation (Tzanetakis and Cook, 2002).

Gjerdingen and Perrott (2008) have shown that humans are remarkably good at genre classification. As a matter of fact, humans can accurately predict a musical genre based on 250 milliseconds of audio. This finding suggests that we can differentiate genres using only the musical surface without constructing any higher level theoretic descriptions. Therefore, it is not surprising that most of genre classification for digitally available music had been done manually until recently. Because of a surge in quantity of produced songs, automatic genre tagging would be beneficial for both music streaming services and users.

Most of the approaches to genre classification imply using audio features. We chose to work only with songs lyrics while tackling this problem. Our motivation lies in faster processing of text when compared to audio analysis. Additionally, audio files can be hard to procure in wanted formats and they are often substantially larger in size than lyrics as well. Intrigued by a visual essay on repetitiveness in pop lyrics (Morris, 2017), we decided to further explore the importance of repetition in song lyrics. Simple statistical analysis can show that the music is becoming more and more repetitive, especially pop songs. Interestingly enough, this trend is particularly obvious and more intense in top 10 songs on charts throughout the years.

Since it is generally much harder to classify genres using solely lyrics, our main goal was to achieve results that can rank with audio-based classifiers in which we eventually succeeded. In order to do so, we tested the performance on shallow models such as Support Vector Machine (SVM) and Logistic Regression with various feature sets. Among them was a set based on TF-IDF scheme and *fastText* library which we used for learning word embeddings. Described model was also utilized for comparison with repetitiveness feature model on which we put a special focus. Both of our models heavily outperformed the used baseline model.

We decided to implement consensus clustering to select specific features for lyrics repetition. It represents a proxy task to estimate the quality of scrutinized features. Quality of clustering using specific features was translated as their importance. Our hypothesis is that the general song clustering task is considerably correlated with genre classification problem, therefore we use it as a feature selector in this fashion. We hope to achieve a more robust estimation since we can use unlabeled data without restricting ourselves to a specific dataset. After constructing and selecting the features, we evaluated all variants and compared the repetition features model with audio classifier baseline.

2. Related Work

Genre classification is dominantly approached by audio analysis. However, regarding lyrics-based methods, most of them involve feature engineering. In more recent works, recurrent neural networks were applied for this task. More precisely, hierarchical attention network was adapted to song lyrics (Tsaptsinos, 2017). This is supported by a hierarchical layer structure that lyrics exhibit - words combine to form lines, lines form segments, and segments form a complete song. Another idea that inspired this project was a clustering method based on consensus. We stumbled upon it in gene expression analysis where it was used as a method for class discovery (Monti et al., 2003). It is depicted as a robust method that can validate the number of clusters. Furthermore, the method deals with the problem of random center initialization.

In our work, we combined several ideas. Consensus clustering was used as a feature selector and also as a general indicator of feature quality. Compression algorithm served to estimate repetitiveness of song lyrics. Besides that, we exploited repetition structures in order to discover a connection between lyrics and accompanying songs' audio features. In the attempt to do so, convolutional networks were utilized to capture patterns in repetition structures.

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 31–35, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

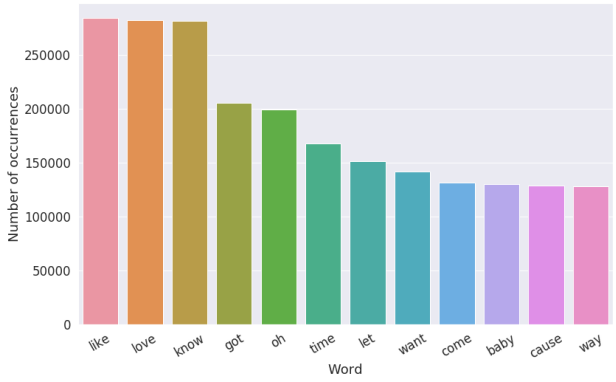


Figure 1: Word frequency on the MetroLyrics dataset

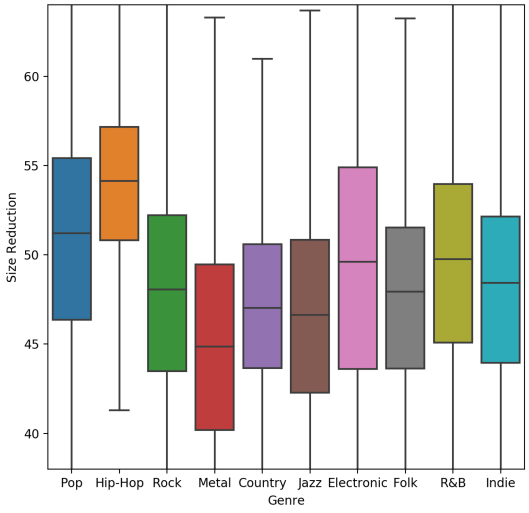


Figure 2: Boxplot of lyrics size reduction by genres

3. Datasets and Preprocessing

Since building a custom dataset was not feasible due to time restrictions, we decided to use *380.000+ lyrics from MetroLyrics* dataset from Kaggle. The dataset contains song lyrics ranging from year 1970 up to 2016, along with metadata such as *song name*, *genre* and *artist name*. The raw dataset is multilingual, although the dominant language is English. To filter out non-English content, we used statistical language detector based on Naive Bayes classifier[1]. Frequency of the top 12 words is shown in the Figure 1 and it approximately follows a Zipfian structure as expected.

Along with ten basic genres, the dataset contains entries labelled as *Other* and *Not Available*. The decision was to throw out these instances since they do not contribute to the previously depicted objective. The last filtering step includes heuristic filtering based on compressibility of the lyrics. For each song we calculated the compression ratio using the Lempel-Ziv-Welch (LZW) compression algorithm (Welch, 1984).

We manually analyzed the content of the songs with the highest and the lowest compression ratio and decided to throw out all songs with compression ratio less than 0.02. After filtering steps, the dataset contains around 214.000 songs classified into 10 distinct genres. Preprocessing pipeline includes lowercasing and stopwords removal. The lyrical content is tokenized to form words and bigrams before the TF-IDF vectorization step.

Additionally, we conducted experiments on a smaller dataset with close to 50.000 instances. They were extracted from Million Song Dataset[2] as a subset called MusixMatch. This dataset contains lyrics and accompanying audio features which we used for a baseline model.

4. FastText Model

The problem we faced was how to properly vectorize a song so that it can be used by various machine learning algorithms. We wanted to generate a dataset of pairs, containing vector as an instance and genre as a label, to achieve a setup that can be used for training by any supervised algorithm. Vectors would be constructed using the lyrics of a song. As explained in section 3., the lyrical preprocessing had been done before engaging into the model itself. The initial idea was to simply use a word embedding algorithm on the entire song lyrics and then utilize a classifier such as SVM. The word embedding algorithm we chose to use was *fastText* (Bojanowski et al., 2016). It produces a 300-dimensional vector for an input word or n-gram. The results of using the entire song as a vector were abysmal due to its time consuming nature. Therefore, entire lyrics were not translated into the vector space, only the most important parts of it. The measure of "importance" that we used was the TF-IDF weighting scheme. Next step is to find the top 10 pairs consisting of weight and n-gram, and calculate the final song vector simply by reweighting the top 10 n-gram vectors by their TF-IDF scores and aggregating them.

This approach also leaves room for improvement since incorporating additional features is easy to do, merely by concatenating the 300-dimensional lyrics vector with any other numeric feature. Ones that have have a positive effect on the performances (ablation study may be considered) can be kept in the final model.

5. Repetition Model

Humans can easily recognize a repetitive song, but it is a difficult task for the computer because it lacks an appropriate measure. Using a compression algorithm to estimate repetitiveness of lyrics was proposed in (Morris, 2017). We tried this method on our dataset. Inspired by the sufficient differences in compressibility of lyrics by genres, we saw an opportunity to explore the possibility of informational gain in such phenomenon. Figure 2 shows a variation of this measure, size reduction of compressed lyrics in percentages, displayed on vertical axis. It is calculated as $1 - r_c$, where r_c is compression rate i.e., ratio of the size of compressed and uncompressed lyrics. *Hip-hop* genre noticeably departs from other ones. This can be supported by

[1] `https://code.google.com/archive/p/language-detection/`

[2] `http://millionsongdataset.com/`

the specificity of hip-hop lyrics as they are often the longest and can have intricate repetition structures.

5.1. Consensus Clustering

Traditional clustering algorithms suffer from several problems: random initialization of centers at the beginning can introduce a certain bias in results and moreover, validation of the number of clusters can pose an obstacle. Consensus clustering approaches mentioned problems as a method based on resampling. Predictions of a chosen clustering algorithm on a subsample (e.g., random 80% of data) are stored in multiple runs. In the end, for each entry (i, j), the calculated consensus matrix records the number of times items i and j are assigned to the same cluster divided by the total count of both items being selected (Monti et al., 2003). Let $D^{(1)}, D^{(2)}, \ldots, D^{(H)}$ be the list of H perturbed datasets, where H is the number of iterations. Entries of the connectivity matrix for each iteration $h \in \{1, 2, \ldots, H\}$ can be formulated as:

$$M^{(h)}(i,j) = \begin{cases} 1 & \text{if items } i \text{ and } j \text{ are in the same cluster,} \\ 0 & \text{otherwise.} \end{cases}$$

Finally, let $I^{(h)}$ be the indicator matrix such that the entry at position (i, j) is equal to 1 if both items are present in $D^{(h)}$, and 0 otherwise. The consensus matrix $\mathbf{C}$ can be denoted as follows:

$$\mathbf{C} = \frac{\sum_h M^{(h)}(i,j)}{\sum_h I^{(h)}(i,j)}.$$

In the final step, a standard clustering algorithm should be used once again to complete the clustering based on the calculated consensus matrix. Entries of the matrix $\mathbf{C}$ are now interpreted as similarity measures of a given pair.

5.2. Consensus Clustering as a Feature Selector

When crafting features for a machine learning task, one may be unsettled by the problem of estimating their importance i.e., quality. We chose a path of evaluating the results of clustering with features "on trial", hypothesizing that models which produce finely distributed clusters will also work well on the genre classification problem. This approach was used instead of standard feature selection methods to achieve a more robust estimation.

By introducing consensus clustering, we have a luxury of analyzing the resulting consensus matrix for each feature that is tested. The key to evaluate how well data was clustered lies in consensus distribution. Ideally, all consensus values should be either 1 or 0. This means that the algorithm was pretty decisive in every iteration and consistent throughout them. The best fit for number of clusters K can be found by studying the cumulative distribution function (CDF) of consensus which can be formulated as:

$$CDF(x) = \frac{\sum_{i<j} \mathbf{1}\{\mathbf{C}(i,j) \leq x\}}{N(N-1)},$$

where N denotes the number of rows (or columns since matrix $\mathbf{C}$ is symmetric). By inspecting the CDF shape and area under the curve, its bimodality can be assessed, which suggests the presence of clusters (Monti et al., 2003). The goal is to find the largest K that induces a large enough increase in the area under the CDF. We decided to pick K with the greatest area increase. In the final assessment of a feature, area under the CDF and its corresponding increase for the best fitted K are interpreted as their measurement of importance.

In the exhaustive process of crafting and selecting features that consider repetitiveness of song lyrics, we found that the LZW compression algorithm provides the best results. More specifically, the ratio of compressed lyrics size with LZW to original lyrics length was calculated. Other compression algorithms such as Huffman and LZ77 produced subpar scores when compared to LZW. Another idea was to create a matrix of co-occurrences. Entry at the position (i, j) is 1 if i-th word is the same as the j-th word, and 0 otherwise. Removal of punctuation and odd characters preceded the calculation. Lemmatization of given lyrics turned out to produce slightly better results, therefore it was kept in the final model. We tried to capture the structure of repetition within the co-occurrence matrix with minimal information loss. Standard matrix norms and custom made ones turned out to be poor choice for this task, so we tried a different approach described in the following subsection.

5.3. Approach by Convolutional Networks

Going one step further, we utilized a convolutional neural network (CNN) to recognize regularities in repetition structure of a specific genre. Co-occurrence matrix for each song described in the previous subsection is multiplied by the corresponding compression ratio and provided as an input to CNN. We padded the matrix with zeros to standardize its size to 1024×1024 (or trimmed it if it was larger), considering that the average word count of lyrics in used datasets was close to 1000. The architecture of the CNN was modeled on AlexNet (Krizhevsky et al., 2012). A simpler version was constructed with adapted number of parameters, primarily in the input layer. The training was enabled by adding a final softmax layer that was used to predict genre labels. Dimensionality of the output layer is changed correspondingly to the number of different genres in the given dataset.

The intuition behind this model is to connect certain patterns in repetition with rhythmic structure of a song. In other words, we presume that such correlation will be of utmost importance in genre classification task.

6. Results and Analysis

In the evaluation of results on MetroLyrics dataset, Natural Language Toolkit (NLTK) was used to provide a baseline model. Specifically, we chose the Multinomial Logistic Regression (MLR) as a classifier on top of a plain TF-IDF Vectorizer. Our models were evaluated using SVM classifiers. Test set contained 25% of the whole dataset which adds up to roughly 50.000 instances. Both the Repetition and the FastText model outperformed the baseline classifier. We supported this with t-tests (10-fold cross-validation) grounded on the micro F_1 scores. Test resulted in statistically significant difference with the significance level of 0.01. Performance scores are depicted in Table 1.

Table 1: Model performances on MetroLyrics Dataset

Model	micro-F_1
NLTK	0.192
Repetition Model	0.379
FastText Model	0.531

Regarding the comparison of our models, we showed that the FastText Model performs better than Repetition Model on MetroLyrics dataset with the same statistical setup. This is quite intuitive considering the larger informational gain of semantics in words. Nevertheless, the surprising fact is the ability of Repetition Model to cope even with plentiful genre types, which is 10 in this case.

We wanted to compare our Repetition Model with a system which is based on audio features. Implying hypothesis is that sound intuitively contains more information about genres than the lyrics. Custom model was built for that purpose. It bases its logic on features such as mel-frequency cepstral coefficients (Logan, 2000), spectral components (e.g., spectral rolloff, chroma, spectral contrast, etc.), tempo... *Librosa* library was utilized for extraction of mentioned features with songs' corresponding audio files provided as input. Calculated values are concatenated to form a numeric vector which is then forwarded to the classifier. Most of these features are widely used nowadays in various machine learning problems in the field of audio analysis (Darji, 2017).

Table 2 shows the comparison of the described baseline to Repetition Model by displaying their respective micro and macro F_1 measures. MLR was used as a final classifier to predict genres on test set (10.000 instances) of the MusixMatch dataset which contains four different genre types. The presumption that our Repetition Model can rank with the audio-based model in genre classification task is supported by the achieved results. It is hinted that the models produce roughly the same scores. We leave further experimenting with repetition structures as an opening to future work. Also, using larger corpora for training our models would very likely result in better performances. This behaviour was noticed when we experimented with training set sizes.

Repetition model performed extremely well in *hip-hop* songs. This hints how hip-hop songs are quite distinctive in their lyrical structure. We also noticed similarities between *pop* and *R&B* songs, which can be deducted from the confusion matrix displayed in Figure 3. Finally, constructed models provided decent results on used datasets and turned out to be a satisfying choice in classifying genres for lyrically dominant songs. It is due to notice the flaw of this approach, considering the complete inability to distinguish genre of purely instrumental songs. Despite this fact, it is not a worrying factor because of the abundance of lyrical songs which results in them largely outnumbering the instrumental ones.

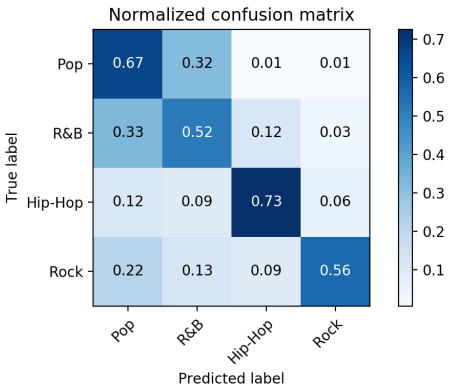


Figure 3: Normalized confusion matrix on MusixMatch dataset of the Repetition Model

Table 2: Audio Baseline and Repetition Model results on MusixMatch Dataset

Model	micro-F_1	macro-F_1
Audio Baseline	0.658	0.601
Repetition Model	0.662	0.609

7. Conclusion

Genre classification is an increasingly approached problem, mostly due to the fact of data explosion i.e., growth of online music databases. Though it is often tackled with audio analysis, some benefits like faster processing and easier access arise in studying the songs' lyrics. We presented two different models, achieving the best results with the FastText Model, but also gathering interesting insights of lyrical repetition by analyzing the Repetition Model.

The more usual approach depicted within the FastText Model turned out to have best performances. This is not a surprising fact since larger informational gain is available in the semantics of song lyrics than in the repetition structure itself. Nevertheless, grasping the phenomenon of repetitiveness in lyrics could prove to be very useful because of the growth of repetition both in quantity and importance. It can especially prosper in the analysis of genres that have strong bonds with lyrical structure.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Mittal Darji. 2017. Audio signal processing: A review of audio signal classification features. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2:227–230, 05.

Robert O. Gjerdingen and David Perrott. 2008. Scanning the dial: The rapid recognition of music genres. *Journal of New Music Research*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolu-

tional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

Beth Logan. 2000. Mel frequency cepstral coefficients for music modeling. In *In International Symposium on Music Information Retrieval*.

Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. 2003. Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Machine Learning*, 52(1):91–118, Jul.

Colin Morris. 2017. Are pop lyrics getting more repetitive? *The Pudding*.

Alexandros Tsaptsinos. 2017. Lyrics-based genre classification using a hierarchical attention network. *18th International Society for Music Information Retrieval Conference*.

George Tzanetakis and Perry Cook. 2002. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10:293–302.

Terry A. Welch. 1984. A technique for high-performance data compression. *IEEE Computer*, pages 8–19.

Detecting Emotion from Text in Context with the Help of Emojis

Ivana Kinder, Natko Kraševac, Filip Serdarušić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ivana.kinder, natko.krasevac, filip.serdarusic}@fer.hr

Abstract

A person's feelings are a part of their inner, mental experience. They are tied to the individual's past experiences, the associations they have acquired through life. Emotions, on the other hand, are a physical response to stimulus and their expression is highly universal, making them highly recognizable in human interaction. This is crucial for human coexistence and effective communication. In textual form, we can see what someone said, but not the way that it was said, which plays a substantial role in what they actually meant. In the context of NLP, emotion detection from text presents a greater challenge than that of sentiment detection, which has been highly extensively studied. In this paper, we approach detection of emotion from text in context within the frame of three turns of conversation in textual messages. This type of text is specific in the sense that it is has many errors, irregularities and deviations from the standard language, but it also often includes the use of emojis, which can provide a nonverbal aid in conveying emotions. In this paper, we explore how this can be helpful in the task of emotion detection.

1. Introduction

Detecting emotions in verbal interactions is something that humans do automatically. Alongside the verbal messages that are conveyed, while communicating, humans express emotions with many other nonverbal cues such as tone of voice, facial expressions and other more ambiguous body language. Humans can aggregate that information, and with the help of their general knowledge and experience, automatically and in most cases, correctly identify the emotion that is being conveyed. In the case of communicating through text, the messages are stripped of all of the very useful nonverbal information and emotion detection becomes significantly more challenging, for humans as well as machines.

In the case of observing a single utterance such as *"Are you kidding me?"*, one cannot be sure of whether the person writing it is angry, or annoyed with the person to whom they are speaking, or even happy and excited about something the other person has told them. This is where looking into the context of this utterance can shed further light onto what the person is feeling. In this paper, we use a dataset of textual conversations that consist of a user utterance along with two turns of context, as shown in Figure 1. These are text messages, so they have misspelled words, abbreviations, inconsistent use of capitalization and punctuation, all of which make their processing more challenging. On the other hand, these text messages also include emojis, which we have found to be of great value in the task of detecting emotion from text. In this paper, we implement a BiLSTM based model for emotion detection from text in context, we explore the significance of emojis in this task and we explore several ways of incorporating those emojis into our model. We evaluate those approaches by their impact on the final classification score.

2. Related Work

2.1. Emotion Detection

Many previous approaches to emotion detection from text were based on rules such as (Hutto and Gilbert, 2014) where authors classified text based on several rules such as using capitalization, punctuation, frequency and adjectives. Haddi et al. (2013) show the importance of text preprocessing such as stemming, removing stopwords or cleaning the text of HTML tags.

2.2. Use of Emojis in NLP

It has been shown by Novak et al. (2015) that emojis are highly indicative of emotion and sentiment. There have been many studies on how emojis impact the sentiment and emotion detection in text. Various approaches were proposed on how emojis should be represented.

Eisner et al. (2016) in their *emoji2vec* paper have released 1661 emoji embeddings trained directly on unicode descriptions of emojis. Embeddings have been created by summation of individual word vectors found in Google News *word2vec* that are describing the particular emoji and trained in a classic word2vec manner. The resulting emoji embeddings have proved useful in social NLP tasks where emojis are frequently used (e.g. Twitter, Instagram. etc.).

Felbo et al. (2017) have shown how the millions of texts on social media with emojis can be used for pretraining models and thereby allowing them to learn representaions of emotional content in texts. The result was pretrained DeepMoji model presenting the state-of-the-art performance on 8 benchmark datasets within sentiment, emotion and sarcasm detection.

Wijeratne et al. (2016) presented the release of EmojiNet, the largest machine-readable emoji sense inventory that links unicode emoji representations to their English meanings extracted from the Web. The extensive database of emoji senses enables the potential integration of emoji with practical and theoretical NLP analysis.

In conclusion, it has been shown that using emojis in sentiment or sarcasm detection in social media text has been useful, and often crucial for improving the performance of existing models.

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 36–39, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

3. Data

The dataset was obtained from the SemEval-2019 Task 3: EmoContext Contextual Emotion Detection in Text competition. In the task, textual dialogue is given. A sample consists of a user utterance along with two turns of context. The emotion of the user utterance is classified into one of three emotion classes: *Happy*, *Sad*, *Angry* or *Others*.

The training dataset, as shown in Figure 1, is a text file containing 5 columns:

1. A unique number to identify each training sample
2. The first turn in the conversation, written by User 1
3. The second turn of the conversation, written by User 2
4. The third turn, which is written by User 1 as a reply to the second turn
5. The human judged label of the emotion of the third turn of the conversation, based on all three turns. It is either: *Happy*, *Sad*, *Angry* or *Others*.

id	turn1	turn2	turn3	label
24	Bcoz u dont know wat is to miss someone	but sometimes one can't express the same	😪	sad
140	How about you	tired of life or just your day? Aha I'm happy today, thanks for asking	Wow great..!	happy
33	You are fool	So I've been told.	You are dumb	angry
35	Abt me	Can you be more specific?	I want to propose a girl	others

Figure 1: Four examples from the dataset.

The training set consists of 30160 samples: about 5k samples each from *Angry*, *Sad* and *Happy* class and 15k samples from the *Others* class. On the other hand, two test sets are provided, and both have a realistic distribution, as shown in Figure 2: about 4 % each of *Angry*, *Sad* and *Happy* class, while the rest is the *Others* class. Test1 has 2755 samples and Test2 has 5509 samples.

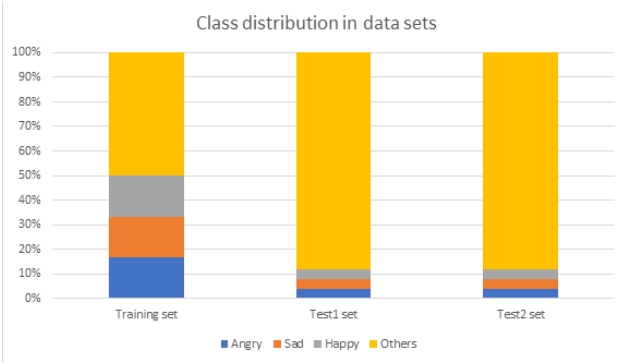


Figure 2: Distribution of classes in the training set and test sets.

Emotion class labeling of these samples was done by 50 trained human judges. Every dialogue was judged by 7 different judges and the majority label was assigned. A Kappa score of 0.58 was observed on the training data set, while a score of 0.59 was observed on the test sets, which shows that there were disagreements of human judges as to what the emotion of a conversation truly is. This goes along with the intuition that recognizing emotions in general has an innate subjectivity to it.

4. Baseline Model

The model which was used as a baseline and then built upon was included in a starting kit provided by the organizers of the SemEval-2019 Task 3: EmoContext Contextual Emotion Detection in Text competition. This section will include a brief explanation of the baseline model.

First, minimal preprocessing is done. Multiple punctuation marks are replaced with single punctuation and white-spaces are added around such punctuation. Duplicate spaces are also removed, and the three turns of conversation are concatenated using an <eos> token.

The emotion detection task is modeled as a multi-class classification problem where given a dialogue, the model outputs probabilities of it belonging to the four output classes: *Happy*, *Sad*, *Angry* and *Others*. The concatenated conversation is then transformed into a continuous vector representation using pretrained 100-dimensional GloVe embeddings. These embeddings are used as input to a plain LSTM layer, which gives a 128-dimensional representation of the sentence. The output of the LSTM layer is then used as the input to the final fully connected layer with four neurons of which inputs correspond to the four classes.

5. Our Approach

The main aim of this paper is to investigate how using emojis in sentiment classification impacts the final classification score. We propose several methods of handling emojis in text.

5.1. Handling Emojis

In this paper we propose, test and evaluate several approaches on handling emojis regarding emotion detection.

In the first, and most simple method, we remove any emojis from the dataset. It will be shown that when using this approach, we lose the valuable information that emojis carry. Our second approach to handling emojis was the use of *emoji2vec* embeddings proposed by Eisner et al. (2016). The third method involved replacing emojis with their corresponding textual description from *emoji* Python library. The description was also processed to gain usability in our classification model.

5.2. Model Description

In the following sections, we explain the model used in our experiments.

Model used in this paper is based on Bidirectional LSTM architecture. BiLSTMs are a logical choice considering they achieve high scores on various NLP tasks, including sentiment analysis. Our model consists of an embedding layer for which we used GloVe embeddings pre-trained on *Twitter* dataset. It is important to understand that in this work we use social media text which is often flooded with grammatically incorrect words and sentences. GloVe Twitter embeddings ensure that we have meaningful representations of such words.

Table 1: Results on test set

Approach	Accuracy	Precision	Recall	F1 score
Without emoji	0.8221	0.4397	0.6958	0.5389
Using emoji2vec embeddings	**0.8659**	**0.5248**	0.7660	0.6229
Using word descriptions of emojis	0.8586	0.5194	**0.8091**	**0.6326**

5.2.1. Text Preprocessing in Our Model

As mentioned, the dataset is gathered from a text messaging application. Preprocessing of the given text can be of crucial importance to model accuracy.

Repeating characters and typing style can be really indicative of sentiment. Because of this, if in conversation we encounter words with extensively repeated characters, e.g. *sweeeeeeeeeeet*, we concatenate a <elong> tag to the conversation in question. Similarly, if we encounter words that are upper-case only, e.g. *WHAT ARE YOU DOING?*, we concatenate a <allcaps> token. Finally, for text with repeating punctuation such as *Why???????* we concatenate a <repeat>. To our best knowledge, by introducing these special tokens, we enrich the original text with additional information otherwise not captured by pure word embeddings. After adding special tokens, we convert the whole text to lowercase, making words easier to match in GloVe embeddings table.

5.2.2. Description of Our Model

As mentioned, BiLSTMs constantly achieve quality results. Wang et al. (2016) effectively use LSTM architecture on a sentiment classification problem. Inspired by that, we base our model on bidirectional LSTM.

Firstly, we use GloVe word embeddings pretrained on Twitter corpus to map words to their continuous vector representation. Words not available in aforementioned set are given a zero-vector representation with the exception of <*eos*> and <elong> tokens that were given a randomly generated vector representation so that we give our model the understanding of these important tokens. Upon these vectors we build an embedding matrix which is used as an input to a bidirectional LSTM with 128 cells. BiLSTM output is then fed into a fully connected layer with number of neurons corresponding to number of classes in our classification problem. The output of the fully connected layer is then put through a softmax activation function giving us a probabilistic representation of a single conversation instance belonging to each of the classes. Figure 3 shows the aforementioned model.

Figure 3: Architecture of our model

6. Evaluation and Results

We have trained the model on 30160 samples in train dateset for 6 epochs using the Adam optimizer with cross-entropy as the loss function. Hyperparameters such as learning rate, batch size, dropout, etc. were optimized using the grid search method on *dev* dataset provided by EmoContext organizers.

6.1. Evaluation Metric

Evaluation was conducted on the predicted class of each sample in the test set by the micro-averaged F1 score of the three emotion classes: *Angry*, *Sad* and *Happy*. The metric is defined as follows:

$$P_\mu = \frac{\sum \text{TP}_i}{\sum \text{TP}_i + \text{FP}_i}, \quad \forall i \in \{Happy, Sad, Angry\} \quad (1)$$

$$R_\mu = \frac{\sum \text{TP}_i}{\sum \text{TP}_i + \text{FN}_i}, \quad \forall i \in \{Happy, Sad, Angry\}, \quad (2)$$

where TP_i is the number of correctly predicted samples of class *i*, while FN_i is the number of samples that do belong to class *i*, but were classified otherwise, and the FP_i is the number of samples that do not belong to class i but were classified as class *i*. The final F1 score is obtained by calculating the harmonic mean of precision P_μ (1) and recall R_μ (2):

$$F1_\mu = 2 \cdot \frac{P_\mu \cdot R_\mu}{P_\mu + R_\mu}$$

Table 2: P, R and F1 scores for each class except *Other* class

Class	Precision	Recall	F1
Angry	**0.530**	**0.868**	**0.658**
Sad	0.495	0.750	0.596
Happy	0.530	0.800	0.637

6.2. Results

The results on test dataset for different approaches regarding handling emojis are shown in Table 1. It is evident that use of emojis significantly improves overall performance of the model. The two approaches (using emoji2vec and

using word descriptions of emojis) achieve similar results, although the latter achieves significantly better micro recall score. In Table 2 it is shown that the *Angry* class results beat the other two classes in recall and $F1_\mu$ score. Fox et al. (2000) conducted several experiments and came with the conclusion that angry or angry/sad emotions are more easily and effectively recognized over other emotion when it comes to facial expressions, but in our understanding it can also be the case with text expressions since angry people tend to be expressive overall.

7. Conclusion and Future Work

In this paper, we experimented with several approaches regarding handling emojis in a emotion classification task. We've shown the benefits of using emoji vector representations over ignoring emojis completely.

With or without emoji, emotion detection from text is still not a trivial task to solve, even with the current deep learning approaches. Currently, huge corpora is available with the growth of social media, but annotation is still time and money expensive so it will be interesting to see how semisupervised or unsupervised approaches compare to current state-of-the-art models. It is evident, even on the dataset used in our experiments, that uneven distribution of examples in classes cause problems on minority classes in form of low precision score.

In future works, we expect that current problems in sentiment analysis, such as negation and sarcasm detection, will be tackled and solved using deep learning approaches. However, there is still a lot of research to be done to get computers to human-level emotion detection.

References

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

Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. 2017. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *arXiv preprint arXiv:1708.00524*.

Elaine Fox, Victoria Lester, Riccardo Russo, RJ Bowles, Alessio Pichler, and Kevin Dutton. 2000. Facial expressions of emotion: Are angry faces detected more efficiently? *Cognition & emotion*, 14(1):61–92.

Emma Haddi, Xiaohui Liu, and Yong Shi. 2013. The role of text pre-processing in sentiment analysis. *Procedia Computer Science*, 17:26–32.

Clayton J Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth 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):e0144296.

Yequan Wang, Minlie Huang, Li Zhao, et al. 2016. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615.

Sanjaya Wijeratne, Lakshika Balasuriya, Amit Sheth, and Derek Doran. 2016. Emojinet: Building a machine readable sense inventory for emoji. In *International conference on social informatics*, pages 527–541. Springer.

Judging a Book By Its Cover: Predicting Partisanship Using Only Article Titles

Mihael Liskij, Dominik Prester, Ivan Šego

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{mihael.liskij,dominik.prester,ivan.sego}@fer.hr

Abstract

Partisan news are a growing problem in today's world. The scientific community has been seeking accurate models which are capable of recognizing false and biased articles to reduce the chance of them causing irreversible damage. In this paper we analyze the problem of predicting partisanship on news articles by only looking at their title to try and possibly improve the speed of detection. To do this, we test two established models by combining them with four features shown to work well in the related field of fake news detection. While we did manage to get an improved performance on both of our models, the raw results show a lot of room for improvement.

1. Introduction

Ubiquitous access to the internet has allowed news and new ideas to spread over the globe with unprecedented speed. This is in many ways positive for society but the last decade has brought to light some fundamental issues with regard to different groups abusing this new medium. The idea of publishing and spreading false and biased information, so called "fake news", is not new, and many news outlets have been used to push a specific agenda long before they moved online, but recent examples from politics around the world have shown that the influence this can have on the democratic process cannot be overstated.

A notable thing about fake news is that it is often hyperpartisan and designed as clickbait. To increase the likelihood that an article gets views, writers often resort to using a catchy title, e.g. "7 tricks that will get your article instantly accepted" or "10 things you must do to get all points on an assignment, number 3 will leave you breathless", just enough to make a reader curious. This, coupled with the ease with which information spreads through social media has caused a lot of concern from many different sides, including the scientific community.

Many researches took up the task of trying to discern between real and fake news using knowledge-based or context-based methods. These methods work well at identifying fake news after publishing, but at that point the damage might have already been done. On the other side, style-based approaches can try to identify fake news at publishing time and thus prevent its spread before it has a chance to happen. Assuming that most partisan news titles have a different enough style from mainstream news titles allows building a much faster system in comparison to systems which look at the whole article.

By trying out two different models with additional features, and comparing them to multiple baselines, we try to gain insights into what works and what does not for predicting partisanship by looking only at the titles, in essence "judging the book by its cover". We theorize that we can get comparable performance to methods which look at the whole article while gaining an advantage in computational performance, but this obviously leaves holes to exploit by authors who mask their titles to look like regular news. Nevertheless, we find that this avenue is worth exploring to see if one could use such a system as a preliminary filter before applying more intensive methods which analyze the article body.

In section 2, we give an overview of existing methods for detecting fake news and partisanship. We go over the different models we decided to evaluate in section 3 while section 4 describes our data set and the method we used for evaluation. The results are presented in section 5 and we finally conclude our paper with section 6.

2. Related work

Various approaches for detecting fake news have been extensively covered in recent work.

Bourgonje et al. (2017) tackle the task of matching the stances of headlines with the stance of the corresponding article bodies. They describe two approaches, one which uses a lemmatisation based n-gram model for the binary classification of headline-article pairs as "related" vs. "unrelated", the other which further classifies the "related" pairs into "agree", "disagree" and "discuss" groups using a 3-class Mallet's Logistic Regression classifier (McCallum, 2002). The combined classifier achieved impressive results, but 25% of the test set unfortunately was not directly applicable to real world scenarios and caused an artificial boost in performance which the authors noted in their analysis.

Mrowca (2017) presented a different solution to the previously mentioned problem of classifying the stance of headline-article pairs into four categories: "agree", "disagree", "discuss", and "unrelated". He used a conditioned Bidirectional Long Short-term Memory (Bi-LSTM) with global and local word embedding features. The model ended up being really good at predicting the relevancy of titles to the article bodies, but had a hard time predicting whether the sentiment was positive or negative.

Potthast et al. (2018) performed extensive experiments on discriminating fake news, hyperpartisan news, and satire based solely on writing style. They also verified, using validation experiments, their finding that the writing style of the left and the right is more similar to each other than it is to the mainstream. Their results show that hyperpartisan news can indeed be distinguished from more neutral

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 40–44, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

news just by using style. The model they used incorporated common features such as n-grams, parts-of-speech tagging, and stop words in addition to news specific feature such as readability scores and dictionary features. Moreover, they applied unmasking as proposed by Koppel et al. (2007) in a novel way for discerning between more broad style categories, such as left-wing vs. right-wing. A similarly extensive exploration of fake news content and how it differs from mainstream content was done by Horne and Adali (2017). By trying out many different features and evaluating their strength for differentiating real news, fake news and satire, they came to important findings, most important of which is, for our purpose, that titles are a strong indicator for distinguishing between fake and real news. Based on their results that fake news titles are usually shorter and use simpler words, they speculate that their writers try to squeeze as much substance into the title, often leaving out stop-words and nouns.

More recently, Colgan and Kakkar (2019) used a multi-feature ensemble to predict hyperpartisan news. The ensemble is comprised of a title, text and link classifier which are then combined using a panel of experts scheme. They used the Semeval hyperpartisan news data set (Kiesel et al., 2018), same one we are using, to train and test their model. In addition to using the data set, they implemented a relabeling scheme to change the labels in hopes of normalizing the data and preventing the model's understanding from getting skewed. Unfortunately, the relabeling scheme turned out to be unsuccessful since their ensemble model generally had worse performance when training on the relabeled data set instead of the original data set.

While previous work was focused on achieving maximum performance by combining title and text features, our focus is on predicting partisanship using only article titles. We would expect this to lead to worse performance on evaluation metrics, but on the other hand, our training times should be much shorter. Our work is done under the assumption that left-wing and right-wing articles have a similar writing style as indicated by Potthast et al. (2018) and that one can use titles to predict relevancy to the text, similar to Mrowca (2017). To try and further improve our models, we will use the best features from Horne and Adali (2017). Even though these features are originally designed for detecting fake news, we assume that they will perform similarly when detecting partisanship.

3. Models

In this section we will give a brief overview of the different models we tried out on the task of predicting partisanship from article titles, the title style features we used for enhancing our models, and the hyper-parameters we used in our training setup.

3.1. Baselines

For the baselines we chose three simple dummy classifiers. The first classifier generates predictions uniformly at random, the second predicts the most frequent label in the training set, and the third classifier generates predictions with respect to the training set's class distribution. Because the data set we will be using is generally balanced, we expect all of the baselines to have a similar performance.

3.2. Bi-LSTM

One of the main models we are evaluating for our task is a Bidirectional Long Short-term Memory Network (Bi-LSTM) (Graves et al., 2005). We decided to use a Bi-LSTM model, even though they are normally used when faced with modeling long-term dependencies in a text, because they achieve good overall performance in various natural language processing tasks. Specifically, a Bi-LSTM, in combination with additional global features, had the best performance when predicting the relevance of titles to news article bodies (Mrowca, 2017).

3.3. SVM

Support Vector Machines (SVMs) have a long tradition in text categorization. They consistently produced state-of-the-art results in text categorization (Joachims, 1998), but lately the natural language processing (NLP) scene seems to have turned more towards neural networks and deep learning. Nevertheless, we decided to try and see if the simpler SVM classifier, which takes as input a dimension reduced matrix of TF-IDF features, works well for our needs.

3.4. Title style features

To further improve all of our models, we will use style features which have been shown to work well for detecting fake news based on titles (Horne and Adali, 2017). These features include: average word length, number of nouns, percentage of stop words, and the Flesh-Kincaid (FK) grade level readability test. The FK readability grade is calculated with the following formula:

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right) + 11.8\left(\frac{\text{total syllables}}{\text{total words}}\right) - 15.59$$

These four features have been shown to be the best ones for identifying fake news. We are interested to see if they can also be applied on the task of identifying partisanship.

To test this we combine the features with our two main models and try to see if we get a significant increase in performance. In the case of SVM, these features are simply appended to the TF-IDF coefficients after normalization. As for the Bi-LSTM model, it is expanded by the addition of a fully-connected layer which takes the current output concatenated with the dense representation of features as input. This expanded model is further referred to as the hybrid model.

3.5. Training details

We train the SVM model on the output of a dimension reduced matrix of TF-IDF features. Text titles are lowercased, tokenized, and stop words are removed as a preprocessing step. The best results on our validation set were achieved for n-grams of length one to three, and max features set to 500. Further, the number of features was reduced to four components by applying singular value decomposition (SVD). It turns out that out of all reasonably possible components, four was the optimal choice. We theorize that this gives us a balance with the 4 features described in section 3.4.

Table 1: Contingency table for Bi-LSTM and Bi-LSTM + features models

	Bi-LSTM + features		
Bi-LSTM	Correct prediction	Incorrect prediction	Total
Correct prediction	289	117	406
Incorrect prediction	75	162	237
Total	364	279	643

Table 2: Contingency table for SVM and SVM + features models

	SVM + features		
SVM	Correct prediction	Incorrect prediction	Total
Correct prediction	289	70	359
Incorrect prediction	34	250	284
Total	323	320	643

For Bi-LSTM training we also lowercase, and tokenize the titles before training our model. Input words are encoded using the GloVe word embeddings (Pennington et al., 2014). The pure, end-to-end, Bi-LSTM model is optimized with RMSprop, whereas the hybrid model is trained using the Adam optimizer. For regularization purposes, we use dropout with 10% at the Bi-LSTM layer for both the input and recurrent step, 20% at the dense layer that follows the custom feature input layer and 30% after the concatenation of the Bi-LSTM and dense feature representations.

4. Evaluation

In this section we describe our training setup, how we evaluate the model, and on which data set we trained on.

4.1. Data set

For evaluating our models, we use the SemEval 2019, Hyperpartisan News Detection data set (Kiesel et al., 2018). The data set is comprised of news articles which have been labeled, depending on their bias, as either "left", "left-center", "center", "right-center" or "right". All labels, except "center", indicate partisanship as given in the data set.

The data set is split into two parts, first of which contains 750000 news articles, exactly half of which are partisan and half of which are not, which have been labeled by overall publisher bias, so called "by-publisher" data set. This means that all articles from the same publisher have the same label which might cause models to overfit and learn the general style of writers who publish for a certain newspaper, instead of learning the styles of partisan articles. Additionally, exactly half of the partisan articles are on the left side of the political spectrum, and half of them are on the right. For convenience, this data is split into a training set (80%, 600000 articles) and a validation set (20%, 150000 articles) where no publishers from the training set appear in the validation set. These labels were taken from BuzzFeed journalists or from MediaBiasFactCheck. The second part contains only articles which have been labeled using crowdsourcing and where a consensus was achieved and is identified "by-article". Hence, this part contains only 645 articles, from which 238 (37%) are partisan and 407 (63%) are not. The inter annotator agreement was not reported with the data set.

4.2. Testing

We will train our models on the training part of the larger, "by-publisher", data set with model selection done using the validation part of the same data set. The official test set from the competition was never released so we decided to use the smaller, more precisely labeled, "by-article" data set as our "gold standard" on which we evaluate all metrics. All models should have the same advantage/disadvantage on the "by-article" data set apart from slightly favoring models with better adaptability to the noise of the larger, broadly labeled, data set. We argue that this inconsistency is justified because we expect this to better mirror the real world performance of the models.

To avoid a memory bottleneck issues, the training is performed on a subset of 50000 samples from the "by-publisher" data set. Out of the multiple neural model architectures tested, the best ones were chosen based on their accuracy performance on a 10000 sample subset of the "by-publisher" labeled validation data. It should be noted that out of the 50000 training samples used, 49849 had non-empty titles and that out of the 10000 samples used for model selection, 9164 were non-empty. Pure Bi-LSTM and hybrid models were also compared to each other regardless of other models in interest of further testing whether the additional features affect the model performance. Same was done with SVM models to determine the effect features had on the performance.

5. Results

In this section we present the results of our comparison between the standard models and models which include the title style features mentioned in section 3.4. We also calculated the confidence intervals for the accuracy and micro F1 measure of all of our models to see what kind of performance we achieved.

5.1. Effect of features

Tables 1 and 2 show the contingency tables of the Bi-LSTM and SVM models. Each table space shows the number of pairs which got assigned the correct/incorrect labels for the test set. Our H_0 hypothesis is that the two models disagree to the same amount, which boils down to checking the hypothesis that the number of (incorrect, correct) pairs is equal to the number of (correct, incorrect) pairs. To do that we employ the McNemar's test with a significance level $\alpha = 0.05$.

For the Bi-LSTM model, we discard our H_0 hypothesis with ($p = 0.003$) in favor of the opposite hypothe-

Table 3: Confidence intervals (CI) for accuracy and F1

Model	Accuracy CI @ 95%	F1 score CI @ 95%
Uniform	[49.8, 50.5]	[55.5, 56.2]
Majority	[36.8, 37.3]	N.A.[1]
Stratified	[49.2, 49.7]	[54.8, 55.4]
Bi-LSTM	[56.1, 56.8]	[67.0, 67.5]
+ features	[63.0, 63.6]	[77.3, 77.7]
SVM	[49.6, 50.3]	[56.0, 56.8]
+ features	[55.4, 56.0]	[54.8, 55.7]

sis that there is significant difference in the disagreement, i.e. the introduction of the features had an effect on our model. The test does not say whether this difference is positive or negative from the perspective of accuracy. Likewise, for the SVM model we discard our H_0 hypothesis with $(p = 0.001)$ in favor of the opposite hypothesis.

5.2. Confidence intervals

Another question we tried to answer is, how well can one predict article partisanship by only looking at the titles. To do that we used bootstrap to calculate the confidence intervals (CIs) with a 95% confidence level for the accuracy and F1 score. We used 100 bootstrap samples with every bootstrap sample containing 1000 test cases randomly sampled, with replacement, from the test data set.

The results of our analysis is shown in Table 3. The uniform and stratified models have a CI around 50% accuracy, whereas the majority classifier has a CI around 37% accuracy that coincides with the class distribution of our test set, $37\% - 63\%$, which explains this result. We should note that the CI would have been around 50% had the class distribution been balanced.

The CIs show that models which use feature show better results than models which do not. Looking at the results, we see that both Bi-LSTM models performed better than their respective counterparts of the SVM model. The SVM model, without extra features, seems to have similar performance to our baselines.

Unfortunately, because of the lack of hardware resources, we did not manage to run any other state-of-the-art model on this particular task, leaving us unable to comment on the relative performance of our Bi-LSTM model. In general, we expect that having the whole data set labeled by article, instead of by publisher, would lead to better results. On the other hand, we suspect that there are several plausible causes for the worse performance of the SVM models relative to Bi-LSTM models. One of them pertains to the fact that the TF-IDF scheme only takes into account word frequencies, while ignoring word order which carries with it more information. The other cause might be that, given the relatively small data set we trained on, the TF-IDF scheme did not learn the representation of all words which might appear in the test set.

[1] Only one class so F1 cannot be computed

6. Conclusion

We analyzed the problem of predicting partisanship from article titles from the perspective of numerous research done on fake news detection. Existing research identified a key set of style features which work well on the task of detecting fake news, and by combining these features with a SVM and Bi-LSTM model, we showed that they can indeed improve the performance of those models.

Future research could focus on exploiting the whole data set, instead of a subset, by using more computing power or optimizing the training process. Another possible research direction could try looking at additional model architectures, such as attention models or models which are based on character embeddings.

References

Peter Bourgonje, Julian Moreno Schneider, and Georg Rehm. 2017. From Clickbait to Fake News Detection: An Approach based on Detecting the Stance of Headlines to Articles. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*, pages 84–89, September.

William Colgan and Keton Kakkar. 2019. Multi-Perspective Ensemble for Hyper-Partisan News Detection. page 6.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2005. Bidirectional lstm networks for improved phoneme classification and recognition. In Włodzisław Duch, Janusz Kacprzyk, Erkki Oja, and Sławomir Zadrożny, editors, *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*, pages 799–804, Berlin, Heidelberg. Springer Berlin Heidelberg.

Benjamin D. Horne and Sibel Adali. 2017. This Just In: Fake News Packs a Lot in Title, Uses Simpler, Repetitive Content in Text Body, More Similar to Satire than Real News. *arXiv:1703.09398 [cs]*, March. arXiv: 1703.09398.

Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, pages 137–142, Berlin, Heidelberg. Springer Berlin Heidelberg.

Johannes Kiesel, Maria Mestre, Rishabh Shukla, Emmanuel Vincent, David Corney, Payam Adineh, Benno Stein, and Martin Potthast. 2018. Data for PAN at SemEval 2019 Task 4: Hyperpartisan News Detection, November. type: dataset.

Moshe Koppel, Jonathan Schler, and Elisheva Bonchek-Dokow. 2007. Measuring Differentiability: Unmasking Pseudonymous Authors. *Journal of Machine Learning Research*, 8(Jun):1261–1276.

Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit.

Damian Mrowca. 2017. Stance Detection for Fake News Identification.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word rep-

resentation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Martin Potthast, Johannes Kiesel, Kevin Reinartz, Janek Bevendorff, and Benno Stein. 2018. A Stylometric Inquiry into Hyperpartisan and Fake News. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 231–240, July.

Combining the Powers of Clustering Affinities in Style Change Detection

Luka Mandić, Fran Milković, Doria Šarić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{luka.mandic, fran.milkovic, doria.saric}@fer.hr

Abstract

A lot of work has been put into style change detection because it presents itself as a promising method for fighting plagiarism and analyzing authorship. The results have improved over the years and are now satisfactory, but determining the exact number of authors in a document still remains unsolved. We tackle this problem on the PAN19 Style Change Detection task. The provided dataset only allows the use of intrinsic detection. Although we're using a very common method – clustering, we introduce novelty by combining various clustering affinity functions. The results show potential in distinguishing between a smaller set (2-3) of writing styles in an article.

1. Introduction

Style change detection is a task that relies primarily on the fact that every author has unique writing habits that cannot be imitated. It can be used to determine how many people contributed to a group assignment and who wrote a certain part of the paper or article, but the most common use is in plagiarism detection. Due to plagiarism being a grave concern in scientific research in general, much work has been put into solving this task. When a document is plagiarised it is, in most cases, somewhat altered. This way the plagiarist attempts to conceal the wrongdoing but often does not succeed because some patterns used in the source of plagiarism often remain intact. This is what plagiarism detection counts on.

In authorship attribution one has the ability to use various author's publications to detect specific writing style of the author which is then used to validate whether the published work is written by the same author or not. On the other hand, intrinsic style change detection relies only on one document - the one in which style change is to be detected. It is evident that distinguishing between different author styles without having their reference collections is a more challenging problem.

In this paper we are describing our approach to this year's PAN Style Change Detection task. The task is to build a system that can determine the number of authors for a given document. Since this task has a history of repeating on PAN workshops, we acknowledge the fact that the previous solutions successfully perform distinction whether the articles have any style change or not (Zlatkova et al., 2018). We therefore decided to focus on determining the exact number of authors in multi-authored documents. Taking insight into Tschuggnal et al. (2017) and Kestemont et al. (2018) we noticed that most previous solutions were either using clustering or applying supervised deep learning methods. We have come to the idea of incorporating a deep learning model into a hierarchical clustering algorithm by using it as a similarity function for a pair of sentences. This way, the sentences are grouped based on their similarity into groups which represent authors. Later on several other similarity measurement functions were combined with our deep model, described in section 4.

2. Related Work

There has been some research done on multi-authored documents by Akiva and Koppel (2012), Akiva and Koppel (2013). The main focus of their work was attributing text segments to a known number of authors. The task of determining the exact number of authors was left for future research. There have also been several works in the closely related field of intrinsic plagiarism detection (IPD). In the case of IPD there is a main author who wrote at least 70% of the considered text document. In opposition to this, in style change detection a document can contain an arbitrary quantity of text per author. In 2017 began a series of tasks related to the topic of style change detection as a part of PAN workshops. The PAN style change detection tasks were, as follows:

- PAN17: Given a document, determine whether it is multi-authored and if yes, find the borders where authors switch. The task of finding the exact positions of the style changes proved to be too demanding. None of the submitted work performed better than a slightly enhanced random baseline (Tschuggnal et al. (2017).

- PAN18: Since the previous task could not be solved accurately the committee decided for a less demanding problem. Given a document, determine whether it is written by one or more authors, i.e. if style changes are present. This task was solved with high accuracy of 0.89 by Zlatkova et al. (2018). They used word embeddings and manually constructed features together with many models in an ensemble, such as deep CNN model, SVM, random forest and AdaBoost trees.

- PAN19: Determine the exact number of authors in a given document. The only solution published so far is by Müller (2019). The approach outperformed last year's top results by two percent. In order to also determine the number of authors in multi-authored documents, clustering was used with a window sliding approach.

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 45–49, University of Zagreb,
Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

3. Data Overview

PAN19 dataset is publicly available and consists of a train and a validation set. [1] There are 2546 articles in the train set and 1272 of them in the validation set. The documents were created by scraping QA forums and combining posts written by different authors about different topics. Three types of labels are included in the dataset – the number of authors, the list of author switches, and a list of author identifiers corresponding to the chunks of text separated by switches. The data is balanced and contains 50% of single-authored documents. The multi-authored documents are as well equally distributed. The documents can be written by up to 5 authors. Certain sentences appear across different documents, so we've come to the conclusion that the data has already been augmented.

When first examining the PAN19 dataset, we tried to manually detect the switches between authors ourselves. We managed to pay attention to only a few features, e.g. sentence length or use of punctuation and abbreviations. It was extremely hard to assess where the switches are located so we even resorted to taking topic into account, which isn't even a stylistic feature and should not have been considered. We did poorly, often completely missing a switch or incorrectly marking something as a switch. This showed us the strength of using a variety of features and implied how semantics might be misleading.

4. Approach

In order to determine the number of authors in a given article, all the sentences were hierarchically clustered in their corresponding "author clusters". The clustering is performed on each article four times with parameter K set to $\{2, 3, 4, 5\}$ – our data only consists of articles written by 1-5 authors. K designates the number of clusters, i.e. authors. The optimal prediction for the number of authors in a given article is chosen as the K with the highest silhouette score.

The strength and novelty of our solution lies in an ensemble combination of different affinity functions that were passed to the clustering algorithm. Affinity function, in our case, is a function that returns a matrix of pairwise distances between all the sentences in an article.

4.1. Syntactic Affinities

Multiple tree kernels were implemented to measure similarity between a pair of sentences based on their syntactic structure. To build such syntactic representations, the dependency tree parser was used. In this way, also assuming that the same author constructs most of his sentences in a similar way, we express to what extent are the two given sentences syntactically similar.

4.1.1. Dependency Relations Affinity

The core of this affinity function is a similarity metric based on Part-of-Speech (henceforth POS) dependency relations of the comparing sentences. The output from our dependency kernel is always limited between 0 and 1. The similarity score is then transformed to a distance indicator by reversing it back to 1-*similarity score*. Therefore, a distance matrix is created for all sentence pairs in a given article by inverting their POS dependency similarity.
After representing each sentence as a list of its dependency relations, the similarity score is computed using our "dependency kernel". It is based on the overlap and position of these syntactic POS dependencies.

We define such dependency similarity as:

$$sim = \sum_{d_A} \sum_{d_B} \alpha * \mathbb{1}[d_A.dep = d_B.dep] + \beta * \mathbb{1}[d_A.start = d_B.start] + \gamma * \mathbb{1}[d_A.end = d_B.end] \quad (1)$$

where d_A and d_B are dependency relations in sentences A and B, respectively. $\mathbb{1}[c]$ is an indicator function that is equal to 1 if c is true and 0 otherwise.

Each dependency relation consists of three key parts: dependency type ($d.dep$), parent node's POS tag ($d.start$), and child node's POS tag ($d.end$). Parent and child node relationship is taken from the dependency tree we previously parse the sentences into. Dependency type can be *nsubj* (nominal subject), *dobj* (direct object), *aux* (auxiliary), *ccomp* (clausal complement), etc. Parameters α, β, and γ are the weights that we join certain parts of a dependency relation. Through optimization on our sentence pair similarity classification we decided for α = 2, β = 3, and γ = 1. The dataset we evaluated this on is further described in section 4.3.

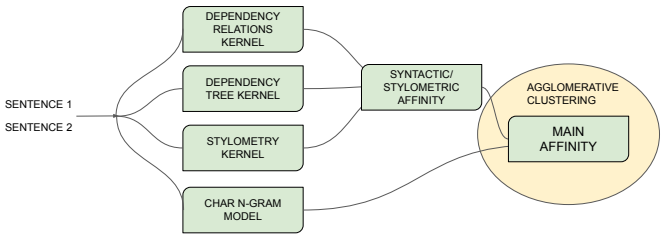


Figure 1: The composition of our affinity functions.

4.1.2. Dependency Tree Affinity

The second affinity function is based on representing each sentence as a list of triplets found in the dependency tree. A triplet is composed of the POS tags of: a node in question, its parent node and one of its children nodes. Unlike the dependency relations affinity, this affinity function doesn't observe the dependencies, only the nodes connected by two consecutive dependencies. Triplets are made for every node in the tree and every node has as many triplets as it has children. In case of a node not having a parent or child (root and leaves), the tags are replaced with the string *'NOTHING'*. POS tags are used instead of actual words to avoid using semantics.

These sentence representations are compared with one another by searching for similar and identical triplets and

[1] https://pan.webis.de/clef19/pan19-web/style-change-detection

adding up the similarity:

$$sim = \sum_{t_A} \sum_{t_B} \alpha * \mathbb{1}[t_A.node = t_B.node] + \beta * \mathbb{1}[t_A.parent = t_B.parent] + \gamma * \mathbb{1}[t_A.child = t_B.child] \quad (2)$$

where t_A and t_B are the POS-tag triplets representing sentences A and B, respectively. Parameters α, β, and γ are the weights joined to certain parts of a triplet similarity and again, through optimization we decided for α = 1, β = 2, and γ = 1. The resulting similarity after normalization is in range from 0 to 1 and is further treated the same way as the similarity of dependency relations.

4.2. Character N-gram Embeddings

By using character *n-grams*, Wieting et al. (2016) have shown that character-based models can successfully be applied to *POS tagging* task which is of great importance for our task. However, their approach tends to learn semantic similarity between shorter phrases. Our task here is different – we aim for the author's stylometric features: function word frequency (e.g. a, if, the), stop-word frequency (e.g. me, you, all, any), special character frequency, etc.

A vocabulary V is defined as the ordered set of N most commonly used char *n-grams* in our dataset. It is important to note that for obtaining a vocabulary, m can be a range of integers. Lower bound of m is m_{min} and upper bound is m_{max}. Every *n-gram* that occurs in a sentence is a one-hot vector of size N. Sentence representation is given by the following formula:

$$\vec{X} = \sum_{m=m_{min}}^{m_{max}} \sum_{i=1}^{n-m+1} \mathbb{1}[x_i^m \in V] W_i^m \quad (3)$$

where x_i^m represents *n-gram* starting at position i of length m and W_i^m is one-hot vector representing *n-gram* x_i^m. As previously noted, word embeddings are commonly learned so that similar words occur in similar contexts. Character *n-gram* embedding is learned in a way that sentences written by the same author should be similar in outer vector space, while those written by different authors should be dissimilar. We use Siamese network architecture, shown on Figure 2. It is commonly used in various NLP tasks such as text similarity measurement (Paul Neculoiu, 2016) and question answering (Arpita Das, 2016).

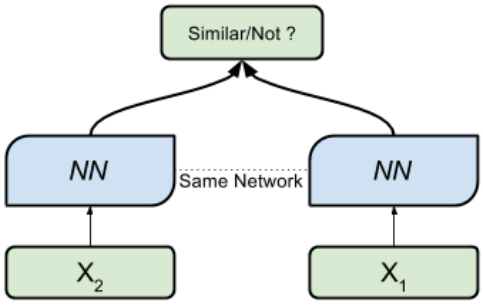


Figure 2: Siamese network architecture

Given a pair of sentences, each sentence is first fed to the neural network (*NN*) model to get sentence embeddings. Those vector representations of sentences are then compared to produce a number between 0 and 1 which represents the similarity score between them, where 1 represents absolute certainty that sentences are from same author and 0 that sentences are from different authors.

We define E_1 and E_2 as the output of *NN* model for first sentence X_1 and second sentence X_2, respectively. Similarity score is denoted by y. We used three different similarity score measurement functions:

- *p-norm* similarity: $y = \frac{1}{1 + \|E_2 - E_1\|_p}$
- *cosine* similarity: $y = \frac{cos + 1}{2}$; $cos = \frac{E_2^T E_1}{\|E_2\|_2 \|E_1\|_2}$
- learned similarity: $y = f(|E_2 - E_1|)$

Since *p-norm* does not have an upper bound, it has to be scaled. This similarity measurement compares how distant the vectors are in outer vector space. The lower the norm of their difference the higher the similarity.

Cosine similarity is in range $[-1, 1]$ so it also must be scaled (in this case, linearly) to the [0, 1] range. It measures cosine of the angle between two vectors and the larger it is, vectors are more similar.

Third similarity score function is actually a *NN* layer which is learning the loss function for sentence embedding. Operator $|\cdot|$ represents element-wise absolute value. This absolute value vector is then fed to the loss layer to obtain a similarity score.

Training is done by backpropagation of the loss gradients through both branches of the *NN* and then taking the mean of gradients as an update step. Training details and results can be seen in section 5.

4.3. Stylometric Features

Since most of the previous related work still involved stylometric feature engineering, to help the clustering algorithm divide sentences into K clusters optimally, it was decided to use such information to combine the strengths of all computed distances. Normally these kinds of features are extracted when sliding window approach is used. Since hierarchical clustering is performed on all the sentences in an article, we had to experiment with such features on sentence level. For some of the commonly used features, it doesn't make sense to use them on just one sentence. A list of final stylometric features is shown in Table 1

We have constructed our own balanced training and evaluation sets from sentence pairs labeled with 1 or 0 depending on whether the sentences come from the same author or not. We ensured that evaluation dataset consists of approximately the same number of negative and positive samples. For both input sentences, we generate a vector of stylometric features. We join the syntactic similarity scores with the vectors and train a neural network on the previously mentioned sentence pairs. The model is a 2-layer fully connected network with 4096 neurons per layer and *tanh* activation functions.

Later on, we combine this model with character n-gram model and the syntactic affinities in the main affinity function by using a 2-layer fully connected neural network. The

Stylometric features
Stopwords ratio
Average word length
Average sentence length (in words)
Verb frequency
Noun frequency
Adjective frequency
Adverb frequency
Special character frequency
Punctuation frequency
Space ratio
Number of noun phrases

Table 1: The final 11 stylometric features we are using to encode the sentences.

affinity function is then used to calculate the 'distance matrix' for all the sentences in the article. To optimize the training time and complexity we define our clustering algorithm with 'precomputed' mode and pass our distance matrix instead of the affinity function. We do the same when evaluating hyperparameter *K* based on the silhouette index.

5. Evaluation

5.1. Character-based Sentence Embedding

Our *NN* architecture for character-based sentence similarity measurement is given in Table 2. To effectively perform convolution, a fully connected layer is put as the first hidden layer to reduce the dimension of the input vector and then it's output is reshaped to a 2D matrix.

Input Sentence
Ngram Layer: size 40000
Fully Connected Layer: size 4096
Reshape Layer: size 64x64
2D Conv Layer 64: size = 11x11, stride = 3x3
Max Pooling Layer: size 2x2
2D Conv Layer 128: size = 7x7, stride = 2x2
Max Pooling Layer: size 2x2
2D Conv Layer 256: size = 3x3, stride = 1x1
Flatten Layer
Fully Connected Layer: size 300

Table 2: Neural network architecture used for character-based sentence embedding

We evaluate our model on the set of sentence pairs with each of the similarity functions (described in 4.2.). This way, without changing the network parameters, we test which similarity score gives optimal performance. Similarity and hyperparameter optimizations are performed on 200 articles for training and 50 articles for validation of the model. Results are shown in Table 3.

Using paired t-test it can be shown that there is no significant difference in using not learned similarity functions (*1-norm* and *cosine*). Learned score function performs slightly better which, we presume, is due to it's ability to adapt and give certain dimensions of output vector more credit in similarity score calculation.

Score function	Accuracy	Std. deviation
1-*norm* score	0.6133	0.02345
cosine score	0.6027	0.02381
learned score	0.6366	0.02369

Table 3: Individual sentence similarity scores

We performed cross-validation on input vector size, embedding vector size and *n-gram* range. As expected, model with the largest values of listed parameters gave the best results but, due to our computational power and memory available, larger values could not be considered. Models with low range of *n-grams* do not involve enough features to learn author stylometric characteristics and those with smaller input vector size do not have the capacity required to do so. We settled with input size vector of 40000, embedding size of 300 and *n-gram* range 2-4.

5.2. Hierarchical Clustering Combining Affinities

After performing clustering for each plausible *K* and choosing the one with the highest silhouette score, we compare our predictions for all the articles with the given number of authors from the dataset. We measure the quality of our solution based on the proposed official metric from the PAN workshop. Due to time limit and computational complexity of our approach, we evaluated our achievements on a limited subset of 50 random articles.

$$\text{rank} = \frac{\text{accuracy} + (1 - \text{OCI})}{2} \tag{4}$$

where OCI is Ordinal Classification Index proposed by Cardoso and Sousa (2011).

accuracy	*OCI*	*rank*
0.28	0.81	0.235

Table 4: Our final results on the number of authors evaluated on the PAN2019 subset.

The proposed system tends to predict 2 or 3 authors. Our features might not have captured underlying compositionality and therefore did not contain enough information to distinguish between so many writing styles.

6. Conclusion

In the task of determining the exact number of authors in a document, incorporating several clustering affinities with deep learning has introduced some novel creative directions for future work. Although this solution doesn't yield significant results, it shows some potential on a smaller set of authors. It might be due to an overly high impact of the syntactic sentence similarity in our approach, which might not be able to capture so many writing styles. We believe one could achieve better results by giving a deep learning model dependency trees instead of defining a similarity metric between them. In this way the model might capture the underlying compositionality. There is also a lot of space for improvement of stylometric features.

References

Navot Akiva and Moshe Koppel. 2012. Identifying distinct components of a multi-author document. In *2012 European Intelligence and Security Informatics Conference*, pages 205–209, Aug.

Navot Akiva and Moshe Koppel. 2013. A generic unsupervised method for decomposing multi-author documents. *JASIST*, 64:2256–2264.

Manoj Chinnakotla Manish Shrivastava Arpita Das, Harish Yenala. 2016. Together we stand: Siamese networks for similar question retrieval.

Jaime S. Cardoso and Ricardo Gamelas Sousa. 2011. Measuring the performance of ordinal classification. *IJPRAI*, 25:1173–1195.

Mike Kestemont, Michael Tschuggnall, Efstathios Stamatatos, Walter Daelemans, Günther Specht, Benno Stein, and Martin Potthast. 2018. Overview of the author identification task at pan-2018: Cross-domain authorship attribution and style change detection. In *CLEF*.

Peter Müller. 2019. Style Change Detection, March.

Mihai Rotaru Paul Neculoiu, Maarten Versteegh. 2016. Learning text similarity with siamese recurrent networks.

Michael Tschuggnal, Efstathios Stamatatos, Ben Verhoeven, Walter Daelemans, Günther Specht, Benno Stein, and Martin Potthast. 2017. Overview of the author identification task at pan-2017: Style breach detection and author clustering. In *CLEF*.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Charagram: Embedding words and sentences via character n-grams.

Dimitrina Zlatkova, Daniel Kopev, Kristiyan Mitov, Atanas Atanasov, Momchil Hardalov, Ivan Koychev, and Preslav Nakov. 2018. An ensemble-rich multi-aspect approachfor robust style change detection.

The Performance of BERT on Implicit Polarity Detection

Andrej Mijić, Gregor Orlić, Luka Abramušić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
andrej.mijic@fer.hr, gregor.orlic@fer.hr, luka.abramusic@fer.hr

Abstract

Detection of implicit sentiment polarity is a quite underdeveloped area of NLP, with most sentiment analysis systems still relying on words with strong sentiment. Recent systems based on language modeling, the latest being BERT, are able to generate advanced sentence representations which may be able to capture implicit sentiment much more accurately than previous models. In this paper we evaluate the performance of BERT's representations on the CLIPEval Implicit Polarity of Events task from SemEval-2015. We achieve superior performance compared to existing models. We also analyze the difference in performance of BERT's representations on different classes of events present in the CLIPEval dataset and find that there is a significant difference between two groups of event classes.

1. Introduction

Sentiment analysis aims to classify the sentiment expressed in a given text into positive, negative or neutral classes, with varying levels of class granulation. A large amount of work on sentiment analysis has been done in the NLP community, mostly using datasets composed of tweets or various Internet reviews.

Particularly tweets and reviews usually consist of words which explicitly express strong sentiment, such as "amazing" or "appalling". For that reason, most traditional sentiment analysis systems rely on such sentiment carrying words to classify the sentiment of an entire document or text. Surprisingly little work has been done on sentiment analysis of sentences or texts that lack such explicit expression of sentiment.

Most existing relevant datasets contain examples which use explicit sentiment carrying words. An important exception is the dataset used in task 9 of the SemEval-2015 competition (Russo et al., 2015), the CLIPEval Implicit Polarity of Events task. We use the same dataset in this paper.

The BERT system has shown remarkable performance on many transfer tasks, including sentiment analysis. The sentiment analysis datasets BERT was trained on, such as the *Stanford Sentiment Treebank* (Socher et al., 2013), use explicit sentiment carrying words as well. We evaluate the performance of BERT on the CLIPEval task. We obtain superior results with BERT compared to other systems used on CLIPEval, as BERT is able to capture the sentence context more accurately.

We also compare BERT's implicit sentiment analysis performance on different classes of events, which are available as part of the CLIPEval competition. We show that BERT exhibits significantly varying performance depending on the class of the event.

2. Related Work

Most existing work on sentiment analysis is focused on using lexical resources. Deriu et al. (2016) and Cliche (2017) use emoticons to infer tweet polarity, enabling them to use large datasets of tweets for training, apart from the one supplied by the SemEval competition. Baziotis et al. (2017) utilize a custom text processor in the preprocessing stage to perform sentiment-aware tokenization which recognizes explicit sentiment expressions.

As for implicit polarity detection, Chen and Chen (2016) use only data without explicit sentiment carrying words. Utilizing both a bag-of-words and a pre-trained word embedding approach for features, they train a linear SVM, an RBF kernel SVM, and a convolutional neural network with those features. In this work, we utilize BERT for generating embeddings for entire sentences.

There were only two participants in the CLIPEval competition, and only one of the systems: SHELLFBK (Dragoni, 2015) has an available system description paper. The SHELLFBK system is based on information retrieval, comparing the frequency of syntactic dependency relations between pairs of words in the query text to all such relations in the training set.

3. CLIPEval Implicit Polarity of Events Task

The CLIPEval competition training data consists of 1280 example sentences. Each sentence is assigned a positive, neutral or negative polarity value. Each sentence is also assigned an event class. There are a total of eight event classes in the dataset: *attending event*, *communication issue*, *money issue*, *legal issue*, *going to places*, *outdoor activity*, *personal care* and *(fear of) physical pain*.

The task consists of two subtasks. Subtask A requires the identification of sentence polarity. Subtask B requires the identification of both the sentence polarity and the event class of the sentence. We propose systems for both subtasks.

4. Our Approach

Existing implicit sentiment analysis systems utilizing approaches based on bag-of-words or word embeddings achieve a lower performance compared to systems for explicit sentiment analysis. Our ideas are motivated by the limited effectiveness of systems based on individual words when explicit sentiment carrying words are lacking.

We argue that correctly classifying implicit sentiment requires a system based on sentence embeddings and trained

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 50–53, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

on a large corpus, since such a system would require being able to learn the context of a sentence.

4.1. Bidirectional Encoder Representations from Transformers

The Bidirectional Encoder Representations from Transformers (BERT) system (Devlin et al., 2018) is a multilayer Transformer encoder based on the original Transformer implementation. For the purposes of this paper, we use the **BERT$_{\text{BASE}}$** model with 12 layers with a hidden size of 768, and a total of 110M parameters.

The BERT model gets its power from its pre-training tasks. The objective of the first pre-training task is training a fully bidirectional language model. For this purpose, the model is trained to predict randomly chosen words in the input sentence which are replaced with a *[MASK]* token. This type of training enables the model to take into account the context of the entire sentence.

The other pre-training task is a Next Sentence Prediction task. Since our task is based on single-sentence inputs, we do not consider this pre-training task to be relevant to the performance of our system.

4.2. Our Systems

Our systems are based on fine-tuning BERT for classification tasks. A single classification layer is added to the output of the BERT model, which adds a negligible number of additional parameters.

For subtask A we fine-tune the **BERT$_{\text{BASE}}$** model for a three-class classification task. The classes used are the polarities of the sentence. All BERT model parameters are fine-tuned, along with the additional classification layer. We use the standard cross-entropy loss function. We use only the data supplied by the SemEval competition for fine-tuning.

For our fine-tuning parameters we use a maximum sequence length of 128, a training batch size of 4, a learning rate of $2 \cdot 10^{-5}$. The fine-tuning is run for 10 epochs.

For subtask B we fine-tune two BERT systems and use them as an ensemble. One of the systems is identical to the one used in subtask A. The second system is fine-tuned for an eight-class classification task to predict the event class of the sentence. The fine-tuning is done using the same parameters as for the subtask A systems.

The motivation for using an ensemble of systems for subtask B is the fact that the performance of a system trained to directly classify both the polarity and event class of the sentence using a total of 24 classes is expected to be very low, due to the very low number of examples in several of the classes.

4.3. Baseline Systems

We use a total of four baselines to evaluate the performance of our system. We use two baselines for each task.

For task A, the first baseline is a simple dummy classifier which returns a random result between three classes, representing sentence polarity. We named this baseline "Baseline 1a". The second baseline is an SVM classifier with the Gaussian kernel available in the `scikit-learn` library (Pedregosa et al., 2011), version 0.20.3 and uses the default parameters. It uses word embeddings from the Spacy model `en_core_web_lg`, version 2.0.0. The Spacy version used is 2.0.12. We named this baseline "Baseline 2a". This model is similar to one of the models employed by (Chen and Chen, 2016).

For task B, the first baseline is an ensemble of two random dummy classifier, the first of which randomly classifies the sentence into three classes, representing sentence polarity, and the second of which randomly classifies the sentence into eight classes, representing the event class of the sentence. We named this "Baseline 1b". The second baseline is an ensemble of two SVM classifiers similar to "Baseline 2a". The first classifier classifies sentence polarity and the second one classifies event class. We named this "Baseline 2b".

5. System Evaluation

There are a total of 1280 training examples. For each task we train ten instances of our system on ten different folds. Each instance uses 1152 examples for the training set and 128 examples for the development set. The development set examples are different for each model. For each instance we train their corresponding baselines using only the 1152 training set examples. We use the differences between our system and the baselines trained on the same training set for statistical significance testing.

We evaluate both systems on their respective tasks. We use the official evaluation script provided by the authors of the competition. The evaluation script uses the official gold standard set, which consists of 371 examples.

5.1. Subtask A Evaluation Results

For task A we report micro-average F1-scores in Table 1. We compare the results of our system with the results achieved by the two systems that took part in the CLIPEval competition and our two baselines. All models have been tested on the official gold standard test data from the CLIPEval competition. We were not able to perform a statistical test for comparison with the SIGMA2320 and SHELLFBK systems, as they are not available online. However, we consider the performance of our system to be better.

We use the matched pair one-sided t-test for testing the significance of the difference in F1-scores between our system and the baseline systems. We use a significance level of 0.01 for all tests. In testing the assumptions of the t-test, we use the Shapiro-Wilk test for testing the normality of differences in F1-scores and Levene's test for testing the homogeneity of variance between F1-scores of both groups.

We find that we cannot reject the null hypothesis of normally distributed differences, as we report a p-value of 0.388 on the Shapiro-Wilk test for the differences in F1-scores between our system and Baseline 1a, and a p-value of 0.029 for the differences in F1-scores between our system and Baseline 2a. Levene's test gives a p-value of 0.312 for the homogeneity of variance between our system and Baseline 1a and a p-value of 0.785 for the homogeneity of variance between our system and Baseline 2a. Therefore, we proceed with the t-test.

The t-test gives a p-value below 0.001 for the difference in F1-scores between our system and each of the baselines. We conclude that our system performs significantly better than the baselines with a confidence level of 99%.

Table 1: Evaluation results for Subtask A: polarity identification.

System	F1-score
SIGMA2320	0.38
SHELLFBK	0.54
Baseline 1a	0.37
Baseline 2a	0.58
Subtask A system	**0.85**

5.2. Subtask B Evaluation Results

For task B we report micro-average F1-scores in Table 2. We compare the results of our system with the results achieved by the single system that took part in the CLIPEval competition and our two baselines. We were not able to perform statistical significance testing for comparison with the SHELLFBK system, but we assume our system's performance to be considerably better.

We use the same method as in subtask B to test the significance of the difference between our system and the baselines. The Shapiro-Wilk test gives a p-value of 0.206 for the normality of differences in F1-scores between our system and Baseline 1b and a p-value of 0.233 for the normality of differences in F1-scores between our system and Baseline 2b. Levene's test gives a p-value of 0.396 for the homogeneity of variance between our system and Baseline 1b and a p-value of 0.213 for the homogeneity of variance between our system and Baseline 2b. We cannot reject the null hypotheses and proceed with the t-test.

The t-test gives a p-value below 0.001 for the difference in F1-scores between our system and each of the baselines. Our system performs significantly better with a confidence level of 99%.

Table 2: Evaluation results for Subtask B: polarity and event class identfication.

System	F1-score
SHELLFBK	0.29
Baseline 1b	0.04
Baseline 2b	0.40
Subtask B system	**0.79**

6. Evaluation of System Performance on Different Event Classes

We give the macro average F1-scores on task A for each of the eight event classes in Table 3. The F1-score given is an average for each event class between the F1-scores for all ten models trained as described in section 5. The models have been tested on the official gold standard test data from the CLIPEval competition. The test data has been split by event class for the purposes of this evaluation.

We use analysis of variance for testing the significance of difference between the system performance on different event classes. We use a significance level of 0.01 for all tests. We use the Shapiro-Wilk test for testing the assumption of normality. We use the Levene test to test the assumption of equal variances of all classes. We find we cannot reject the null hypotheses for any of the classes. Proceeding with analysis of variance, we reject the null hypothesis of equal performance on all classes with a p-value below 0.001.

We use a pairwise posthoc t-test to test the significance of the difference in performance between the classes, using the `scikit-posthocs` libary, version 0.6.1. We find that the classes can be grouped into two major groups in which the performance of the system on each class of the group is significantly different from the performance on every class of the other group. The p-values of the pairwise testing are shown in Table 4. The first group consists of the classes: *attending event*, *money issue*, *legal issue*, *outdoor activity* and *(fear of) physical pain*. The second group consists of the classes: *communication issue*, *going to places* and *personal care*.

Table 3: Comparison of F1-scores of the subtask A system for different event classes.

Event class	F1-score
ATTENDING_EVENT	0.84
COMMUNICATION_ISSUE	0.74
MONEY_ISSUE	0.82
LEGAL_ISSUE	0.83
GOING_TO_PLACES	0.73
OUTDOOR_ACTIVITY	0.84
PERSONAL_CARE	0.76
(FEAR_OF)_PHYSICAL_PAIN	0.88

7. Conclusion

We created a BERT-based system for implicit polarity classification using the SemEval-2015 Task 9 dataset and achieved an F1-score of 0.85, much higher than previous systems. We also created a BERT-based system for subtask B of the same task and achieved an F1-score of 0.79.

We evaluated the performance of our implicit polarity classification system on different event classes and concluded there is a significant difference in performance between two groups of classes. We expect further work to be able to show more precisely the reasons for the performance difference.

8. Acknowledgements

We thank the Youth Research Centre (cro. *Istraživački Centar Mladih*) for providing us with the computational resources required for this approach.

Table 4: P-values of the pairwise t-test of the differences in performance on different classes

	A_E	C_I	M_I	L_I	G_P	O_A	P_C	F_O_P_P
ATTENDING_EVENT	1	<0.001	0.618	0.768	<0.001	0.727	0.003	0.055
COMMUNICATION_ISSUE	<0.001	1	<0.001	<0.001	0.377	<0.001	0.305	<0.001
MONEY_ISSUE	0.618	<0.001	1	0.793	<0.001	0.275	0.001	0.006
LEGAL_ISSUE	0.768	<0.001	0.793	1	<0.001	0.398	<0.001	0.009
GOING_TO_PLACES	<0.001	0.377	<0.001	<0.001	1	<0.001	0.075	<0.001
OUTDOOR_ACTIVITY	0.727	<0.001	0.275	0.398	<0.001	1	<0.001	0.035
PERSONAL_CARE	0.003	0.305	0.001	<0.001	0.075	<0.001	1	<0.001
(FEAR_OF)_PHYSICAL_PAIN	0.055	<0.001	0.006	0.009	<0.001	0.035	<0.001	1

References

Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754, Vancouver, Canada, August. Association for Computational Linguistics.

Huan-Yuan Chen and Hsin-Hsi Chen. 2016. Implicit polarity and implicit aspect recognition in opinion mining. page 20–25.

Mathieu Cliche. 2017. Bb_twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms. page 573–580.

Jan Deriu, Maurice Gonzenbach, Fatih Uzdilli, Aurelien Lucchi, Valeria De Luca, and Martin Jaggi. 2016. Swisscheese at semeval-2016 task 4: Sentiment classification using an ensemble of convolutional neural networks with distant supervision. page 1124–1128.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Mauro Dragoni. 2015. Shellfbk: An information retrieval-based system for multi-domain sentiment analysis. page 502–509.

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.

Irene Russo, Tommaso Caselli, and Carlo Strapparava. 2015. Semeval-2015 task 9: Clipeval implicit polarity of events. page 443–450.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Supplementary material: Recursive deep models for semantic compositionality over a sentiment treebank.

Segment Size Impact on Pairwise Interaction Clustering in Authorship Analysis

Leo Obadić, Luka Lazanja, Fabijan Čorak

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{leo.obadic,luka.lazanja,fabijan.corak}@fer.hr

Abstract

Authorship similarity measurement is a challenging problem, as it requires understanding the stylometric features of input sentences. Our main task is to predict the number of authors who collaborated in composition of a single document. The model we propose uses pairwise segment similarities that are enriched with each author's style. We investigate why segment size should be considered and how it affects performance of the model we used on PAN 2019 Style Change Detection task.

1. Introduction

Given a single document, determining how many authors collaborated on it relates to the text forensic research field and is an open problem without a satisfying solution. What makes this task challenging is a possibility of having a single author with deviating writing style. There have only been a few studies about the related problem of authorship segmentation (Rosen-Zvi et al., 2010; Koppel et al., 2011; Akiva and Koppel, 2013; Akiva and Koppel, 2012) but we're unaware of the same problem.

In this paper, we focus on solving this problem with an unsupervised approach as well by sliding over consecutive segments of text. We hypothesize that segment size influences the capability to distinguish authors in document and investigate how. Proposed model is different from other clustering models because we do not group stylistic features directly, instead we group pairwise segments on combination of segment style representation and its similarity. To choose the appropriate number of clusters k, we calculate the score for each possible k and use our own *ad hoc* heuristic rules.

We present our research in several sections. Section 2 presents what has been done so far and how our model fits in. Following section gives insights in dataset used for evaluation. In Section 4 we propose the model and give its description. Finally, in sections 5 and 6 we present the results and end with a conclusion.

2. Related work

There are several research areas closely related to authorship analysis.

Intrinsic plagiarism detection has been explored by Stamatatos (2009) and Zu Eissen and Stein (2006). Its task is to identify plagiarized sections of the document and it contains one big assumption. If one document is selected as plagiarism, it contains at least 70% text written by one author (Kuznetsov et al., 2016).

Clustering a set of documents according to their authorship has seen previous effort (Layton et al., 2013; Daks and Clark, 2016). Others have tried to solve the same problem in a supervised manner (Graham et al., 2005).

Some authors have tried to segment documents according to a topic (Callan, 1994; Choi, 2000; Rosen-Zvi et al., 2010; Reynar, 1994). This task is easier from the one we consider since they cluster documents based on semantics.

There are researchers that analyzed the problem of clustering topics and authors (Akiva and Koppel, 2012). They assume that each author's work reflects a distinct distribution over topics. In contrast to that, we make no such assumption and do inference solely based on author's style of writing.

Rosen-Zvi et al. (2010) explore document authorship segmentation, but in a supervised manner. The problem is similar to ours since our model predicts the number of authors that collaborated in writing it. Others explored same problem in unsupervised manner (Koppel et al., 2011; Akiva and Koppel, 2013; Akiva and Koppel, 2012) but the main difference is that that they know the number of authors beforehand.

To solve our problem, an intuitive approach is to generate style representations for each sentence and then apply a clustering algorithm over them. Suspecting low representational capacity of a single sentence, we explore how segment size influences model performance. Drawing inspiration from (He and Lin, 2016) and its effective use of fine-grained information, we perform clustering on a combination of style representation and similarity for each segment pair. This way we wanted to map vectors in a different space where it would be easier to locate author-switching.

3. Dataset

We used the dataset provided for PAN 2019 Style Change Detection task [1]. It is based on user posts from various sites of the StackExchange network, covering different topics and containing approximately 300 to 2000 tokens per document. For each document, two files are provided, one with post text and the other one with the number of authors and some other information not relevant to this paper. The number of authors ranges from one to five and the class distribution for documents is $(50\%, 12.5\%, 12.5\%, 12.5\%, 12.5\%)$, 50% being the number of documents written by one author.

[1] https://pan.webis.de/clef19/pan19-web/style-change-detection.html/

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 54–57, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

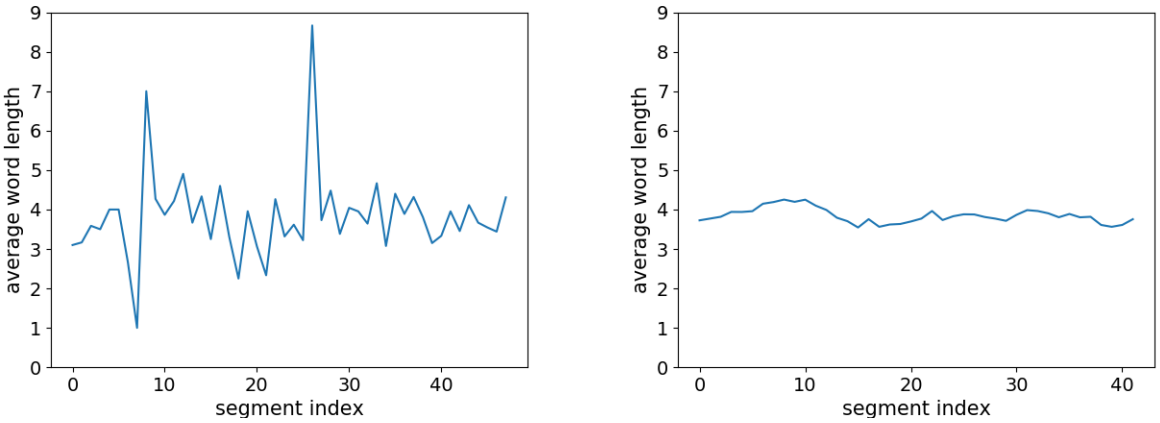


Figure 1: The change of a selected feature (average word length) over successive segments in a single-author document (segment size: left - 1, right - 7)

4. Model description

Our model is composed of three parts:

1. stylometric feature extraction
2. segment interaction modeling
3. clustering

4.1. Stylometric feature extraction

Firstly, we need a way to represent the style of an author with a number. There are many well known stylistic features discovered by linguists. In our model we use same features as Müller (2019) and Zu Eissen and Stein (2006). For average word frequency class we used Google books most common words[2]. The exact 15 features we used are the following:

- Stop-word ratio
- Average word length
- Average sentence length (words)
- Average sentence length (characters)
- Average upper-case characters
- Average word frequency class
- Punctuation frequency
- Yule's K measure
- Function word frequency
- Special character frequency
- Numeric character frequency
- 50 most common POS-tag frequencies
- 50 most common 2-gram word frequency
- Word length distribution (less then 10 letters)
- 50 most common word frequency (w/o stop-words)

As figure 1 demonstrates, features vary over successive sentences, even across single-author documents. However, using a segment beyond that of a single sentence (seven in the right subfigure) mitigates this effect. This motivates for experimenting with segment size[3] and its effect on capturing the style of an author. The issue of a single author's varying feature level is even more pronounced in documents with multiple authors - the model should be able to distinguish between a single author's variation and multiple authors switching. We assume that the model might get increase in accuracy with smart choosing of segment size.

4.2. Segment interaction modeling

We want to separate segments in a way that makes it easy to distinguish the contribution of each author. Having that in mind, we devised a metric M:

$$M(a, b) = a + b + abs(a - b)$$

that is used on each pair of segments for each feature (note that features are not joined in a single vector, we calculate pairwise metric for each feature and present it as feature matrix). An example of a feature matrix for average word length is visualized in Figure 2. It can be noticed that there are distinct regions on similar altitudes. Those regions represent same authors while hillsides represent author switches.

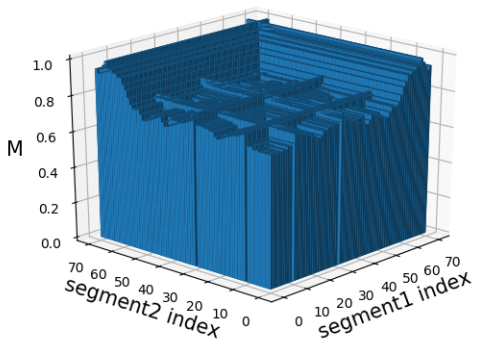


Figure 2: Pairwise segment similarities for one feature.

[2] http://norvig.com/google-books-common-words.txt

[3] number of consecutive sentences

4.3. Clustering

Here we present how the number of authors is chosen according to single features and how this is combined into a final answer.

For each feature matrix we cluster regions of similar altitudes as can be seen in Figure 2. Before that, we remove outliers[4]. Clustering is done with simple k-means ($k \in [1, 6]$) and the final score[5] is saved for each k. We tried predicting appropriate k based on those scores with decision tree but ended up with poor results. Due to big time complexity, we had to discard silhouette analysis. To remedy that, we analyzed scores of randomly sampled documents and came up with heuristic rules (Table 1) that are similar to elbow method. Firstly, we calculate the relative score changes as percentage of score k in score $k-1$. For example, if our score vector is (a, b, c), its relative score changes would be $(\frac{|b-a|}{a}, \frac{|c-b|}{b})$. These relative changes between consecutive scores are referred to as rates. The range of k when performing k-means is $[1, 6]$ because some rules require a right neighbour (hence the upper boundary is 6). The score for $k = 1$ is not considered regarding the first three rules. However, only the values of k from $[1, 5]$ are considered for the final answer. asc_ind denotes the index of the first observed relative increase; max_ind denotes the index of the maximum observed relative change; mean denotes the mean of all relative scores. One of the first three rules is going to trigger if there is an elbow, the fourth rule will trigger if there are no elbows.

Table 1: Heuristic rules to determine k; listed by priority.

	Rule	Vote
1.	first rate is greater than 0.5 mean	2
2.	rates are descending, then start to ascend	$asc\_ind + 2$
3.	left value of max rate is greater than its right value by 30%	max_ind+2
4.	mean is less than 0.45 or rates are descending	1

Heuristic decision making performed on each of the feature matrices yields up to 15 propositions for the optimal k (note that it is possible that no rules will trigger). Final prediction is calculated by performing a hard voting. If there are none propositions, we assume that document is written by single author.

5. Evaluation and results

Our model is evaluated on a test set which consists of 1272 examples. The number of authors and the author distribution of the test set is the same as that of the train set which has been described in section 3. Sentence segmentation has been done with spaCy[6]. For some features, we also did POS tagging, lemmatization or tokenization if required.

[4] removing values deviating from mean by at least 2σ

[5] sum of distances between data samples and their centroids

[6] `https://spacy.io/`

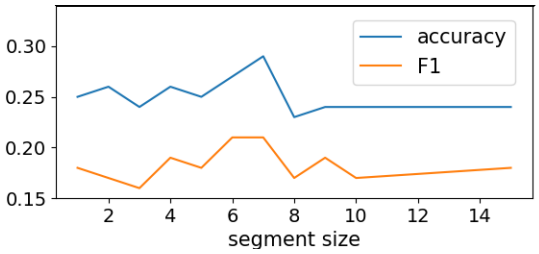


Figure 3: Accuracy and F1 changes with segment size.

We observed how our model performs on different segment sizes (Table 2). Besides the accuracy and macro F1 metrics we used the Ordinal Classification Index (OCI) proposed by Cardoso and Sousa (2011). It is used to measure the error of predicting the number of authors for multi-author documents and yields a value between 0 and 1, where 0 is the best value. The results in Figure 3 show that both accuracy and F1-measure first increase with segment size and after the maximum at 7 they start to decrease. By performing a two-sided t-test on ten folds of the test set, we find the performance with segment size 7 to be significantly better in terms of accuracy than that of segment size 1 (p-value: 0.027) and segment size 15 (p-value: 0.003). We account for the impact of segment size as follows: if it is too small, a single deviation of author's style will have a big impact on the overall segment style and could make the model split one author. On the contrary, if it is too large, it could merge the styles of more than one author into a single one.

Table 2: Prediction results on different segment sizes

Segment size	Acc	F1	OCI
1	0.25	0.18	0.89
2	0.26	0.17	0.92
3	0.24	0.16	0.90
4	0.26	0.19	0.89
5	0.25	0.18	0.89
6	0.27	**0.21**	**0.86**
7	**0.29**	**0.21**	0.88
8	0.23	0.17	0.88
9	0.24	0.19	0.92
10	0.24	0.17	0.88
15	0.23	0.18	0.87
random baseline	0.20	0.17	0.88

We also investigated why exactly did our model perform poorly. As it turns out, the biggest problem was that even if model predicted that there are multiple authors, it could not distinguish the exact number. This is not so surprising since each author can have multiple styles and with increase of authors, it is harder to predict the number of the styles. With everything mentioned in mind, we believe that without better style representation, this problem will stagnate.

Lack of papers regarding this same problem limited our possibility for cross-comparison.

6. Conclusion

In this paper, we discussed the problem of counting the number of authors in a document. We presented an approach that clusters representations of pairwise segment interactions and tested how segment size impacts performance. We observed and accounted for the advantages of a medium-sized segment.

It remains unclear to what extent our approach increases performance, and, if it does, whether it is worth the quadratic time complexity.

Future work should consider more complex ways of pairwise representation: our approach that combines addition with absolute difference could be replaced with one that relies on multiplicative forms. The proposed heuristic rules require further refinement. Other ways of combining the decisions of single features should be considered, such as feeding the similarity matrices to a deep model.

References

Navot Akiva and Moshe Koppel. 2012. Identifying distinct components of a multi-author document. In *2012 European Intelligence and Security Informatics Conference*, pages 205–209. IEEE.

Navot Akiva and Moshe Koppel. 2013. A generic unsupervised method for decomposing multi-author documents. *Journal of the American Society for Information Science and Technology*, 64(11):2256–2264.

James P Callan. 1994. Passage-level evidence in document retrieval. In *SIGIR'94*, pages 302–310. Springer.

Jaime S Cardoso and Ricardo Sousa. 2011. Measuring the performance of ordinal classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(08):1173–1195.

Freddy YY Choi. 2000. Advances in domain independent linear text segmentation. *arXiv preprint cs/0003083*.

Alon Daks and Aidan Clark. 2016. Unsupervised authorial clustering based on syntactic structure. In *Proceedings of the ACL 2016 Student Research Workshop*, pages 114–118.

Neil Graham, Graeme Hirst, and Bhaskara Marthi. 2005. Segmenting documents by stylistic character. *Natural Language Engineering*, 11(4):397–415.

Hua He and Jimmy Lin. 2016. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 937–948.

Moshe Koppel, Navot Akiva, Idan Dershowitz, and Nachum Dershowitz. 2011. Unsupervised decomposition of a document into authorial components. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1356–1364. Association for Computational Linguistics.

Mikhail P Kuznetsov, Anastasia Motrenko, Rita Kuznetsova, and Vadim V Strijov. 2016. Methods for intrinsic plagiarism detection and author diarization. In *CLEF (Working Notes)*, pages 912–919.

Robert Layton, Paul Watters, and Richard Dazeley. 2013. Automated unsupervised authorship analysis using evidence accumulation clustering. *Natural Language Engineering*, 19(1):95–120.

Peter Müller. 2019. Style change detection.

Jeffrey C Reynar. 1994. An automatic method of finding topic boundaries. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 331–333. Association for Computational Linguistics.

Michal Rosen-Zvi, Chaitanya Chemudugunta, Thomas Griffiths, Padhraic Smyth, and Mark Steyvers. 2010. Learning author-topic models from text corpora. *ACM Transactions on Information Systems (TOIS)*, 28(1):4.

Efstathios Stamatatos. 2009. Intrinsic plagiarism detection using character n-gram profiles. *threshold*, 2(1,500).

Sven Meyer Zu Eissen and Benno Stein. 2006. Intrinsic plagiarism detection. In *European Conference on Information Retrieval*, pages 565–569. Springer.

Fact Checking – Just Google It

Marin Sokol, Kristijan Palić, Samir Menaa

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{marin.sokol,kristijan.palic,samir.menaa}@fer.hr

Abstract

Due to massive amount of information available and its questionable verity, we tried to build a model for checking truthfulness of claims. Our model utilizes Google search engine and combines its first-page results with claims from the FEVER dataset. The model classifies each claim as either REFUTED or SUPPORTED. We hypothesize that the textual result returned by Google is enough information for an accurate classification. Even though the FEVER task is closely related, to the best of our knowledge, our dataset and task are unique and this paper provides benchmarks for future studies.

1. Introduction

Given the spreadness and popularity of social media and its importance in our day-to-day life, we are swamped with a massive amount of information. While social media paved the way for free speech it also created the need to verify the information given to us. The easiest and most intuitive way of checking an information today is to simply *Google it*. Googling a claim is free and accessible to every person with an internet connection and does not require any former knowledge about the topic being *googled*. With that in mind, in this paper we tried to simulate that common behaviour by trying to verify a certain claim using the results returned by Google.

For this task we used a random subset of claims taken from the Fact Extraction and VERification dataset (FEVER[1]) presented in (Thorne et al., 2018a), which were then paired with its corresponding Google result. All the claims were either labelled as REFUTED or SUPPORTED depending on their verity.

The main hypothesis of this paper is that any given claim can be supported or refuted based on the textual results found on *Google*'s first page, returned when the entire, unprocessed claim is being searched.

Each claim and its corresponding Google result were embedded into a 768-dimensional vectors using state-of-the-art contextual embedder, BERT, presented in (Devlin et al., 2018). These vectors were then fed into an SVM model whose hyperparameters were then tuned using grid search.

In our preliminary study of only 10,000 claims we obtain comparable results with the best solution submitted to the FEVER Shared Task presented in (Thorne et al., 2018b), even though in Section 4. we argue why the results cannot be directly compared.

2. Related Work

Our task is a subset of the FEVER Shared Task (Thorne et al., 2018a; Thorne et al., 2018b), which required from participants to classify human-written claims into one of three categories: SUPPORTED, REFUTED or NOT ENOUGH INFO, based on the Wikipedia dump provided. Their dataset also contained evidence for each of the (claim, label) pairs on which the classification is based on and required that all models justify their classification by providing sufficient evidence along with the predicted label. An example from their training set is shown in Figure 1.

Claim: Adrienne Bailon is an accountant.
Evidence:
Adrienne Eliza Houghton (nee Bailon ; born October 24 , 1983) is an American singer-songwriter , recording artist , actress , dancer and television personality .
Label: REFUTED

Figure 1: Example taken from the FEVER Shared Task training set. Example contains a claim which is **refuted** using the evidence provided. This evidence is the first line in Wikipedia article for Adrienne Bailon. All of the supporting evidence are lines of articles in the Wikipedia dump provided with the FEVER dataset.

As our task is predicting the label from the corresponding Google search result we are not able to provide the evidence justifying our classifications since our model is not limited to only Wikipedia articles. For the same reason we have decided not to use the claims labelled as NOT ENOUGH INFO. Since our approach utilizes the diverse results found on the web, we hope that it could generalize a lot better and predict labels for claims that Wikipedia-trained models fail on.

It is important to mention that this is not the first approach that takes advantage of Google's advanced search algorithm for solving the FEVER Shared Task. Columbia NLP group, in their approach presented in (Chakrabarty et al., 2018), used Google Custom Search API for retrieving relevant Wikipedia articles for the given claim. They had one of the best evidence recall scores in the entire competition.

3. Approach

This section explains exactly how we created our dataset and shows the end result in Section 3.1. while Section 3.2. defines our model architecture and explains the training procedure.

[1] http://fever.ai/

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 58–61, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

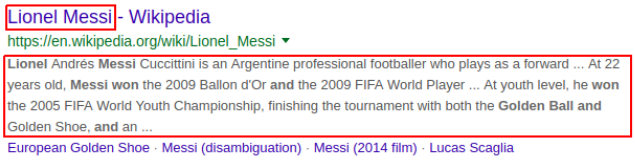


Figure 2: Random Google result which has a website name in its title and which contains multiple "..." sequences inside the body of a result. Both the website name and the "..." were removed from the final result. Red rectangles define the title and body of a result.

Figure 3: Random Google result which title ends with "..." and has a date prepended to its body. Both the "..." sequence and the date were removed from the final result. Red rectangles define the title and body of a result.

3.1. Dataset

From the FEVER train dataset, we randomly sampled 10,000 (claim, label) pairs. We divided them into three sets. The training set contains 7,000 pairs while the validation and the test set each contains 1,500 pairs. All the sets have exactly 73% of claims labelled SUPPORTED and 27% labelled REFUTED.

Each claim was used as a Google query, without any kind of preprocessing. We scraped Google results on the 28th of May in 2019 from 3 different IP addresses located in the United States of America. We only scraped the first page of Google results as this is, in most cases, the only relevant page. From the scraped results we concatenated the title and the body of each returned result as shown in Figures 2 and 3. If the title of the Google result contained the website name divided by the sequence " - " then only the text before the website name would be scraped. This can be seen in Figure 2. If, however, the title of the result ended in the sequence "..." as in Figure 3 then only the text before that sequence would be taken into consideration. In the body of a Google result all occurrences of sequence "..." were removed and if the body started with a date, like for example it does in Figure 3 which starts with sequence "2 hours ago", then that date would be ignored. Titles and bodies were concatenated together with a single whitespace as were all the first-page results returned by Google.

After scraping, we ended up with triplets containing the claim, Google first page result for that claim, and the label. One of this triplets is shown in Figure 4.

3.2. Model

Our model uses state-of-the-art contextual embeddings provided by BERT released in (Devlin et al., 2018). BERT has a way of encoding different segments in the text so that it can compute the embeddings for one text in reference to some other text. That is very useful for problems with separated segments such as Question-Answering tasks in which

Claim: Khal Drogo was introduced in 1996.
Google result:
Khal Drogo Khal Drogo is a fictional character in the A Song of Ice and Fire series of fantasy novels by American author George R. R. Martin and in the first two seasons of its television adaptation, Game of Thrones. **Introduced in 1996's A Game of Thrones, Drogo is a khal**, a leader of the How old are the Game of Thrones characters meant to be? George Game of Thrones: Daenerys Targaryen Fire Immunity, Explained Who gave the dragon eggs to Daenerys? What Happened When Daenerys Was Pregnant With Khal Drogo's Daenerys Targaryen Daenerys Targaryen is a fictional character in George R. R. Martin's A Song of Ice and Fire Introduced in 1996's A Game of Thrones, Daenerys is one of the last two She is forced to marry Dothraki horselord Khal Drogo in exchange for an army for Viserys, who is to return to Westeros and recapture the Iron Throne. Khal Drogo — Revolvy Introduced in 1996's A Game of Thrones, Drogo is a khal, a leader of the Dothraki, a tribe of warriors who roam the continent of Essos. Drogo is portrayed by Khal Drogo Introduced in 1996's A Game of Thrones, Drogo is a khal, a leader of the Dothraki, Drogo is portrayed by Jason Momoa in the HBO television adaptation. Khal Drogo Introduced in 1996's A Game of Thrones , Drogo is a khal, a leader of the Dothraki, a tribe of warriors who roam the continent of Essos. Khal Drogo Khal Drogo is a fictional character in the A Song of Ice and Fire series of fantasy Introduced in 1996's A Game of Thrones, Drogo is a khal, a leader of the Drogo Drogo is a powerful khal or warlord of the fearsome Dothraki nomads. Daenerys Targaryen is promised to him at the beginning of A Song of Ice and Fire. In the A guide to how Daenerys went from Khaleesi to Mad Queen Daenerys comes to very much appreciate Khal Drogo's intensity. When Drogo killed her brother with molten gold, all she could muster up was, Khal Drogo (Game of Thrones character) Khal Drogo is a fictional character in the A Song of Ice and Fire series of fantasy Introduced in 1996's. Who deserved to die more, Khal Drogo or Viserys?
Label: SUPPORTED

Figure 4: Random example taken from our training set. Bold text shows that the given claim can easily be SUPPORTED using the scraped Google result.

BERT takes the question as the first segment and the answer as the second segment.

In our task, BERT takes the claim as the first segment and the returned Google result as the second segment so that it can, in a way, try to prove the claim using the Google result. We use BERT-Base Uncased model which means that we do not make any distinctions between the uppercase and lowercase letters. BERT also uses its own tokenization so no preprocessing on our part is needed. We implemented this embedding procedure by using a Python library provided in (Xiao, 2018)[2] with the maximum sequence length of 512 tokens (if the combination of claim and Google result is bigger than 512 tokens then only the first 512 tokens

[2]https://github.com/hanxiao/bert-as-service

Table 1: This table provides results of our model and the baseline model on the test set.

Model	Accuracy(%)	Balanced Accuracy(%)
Baseline	73.0	50.0
BERT+SVM	**75.0**	**72.3**

are embedded). All the other parameters are the default, recommended values from (Xiao, 2018). The end result for a single (claim, Google result) pair is a 768-dimensional vector.

The 768-dimensional BERT embedding is then used as a feature vector for an SVM classifier. We used kernel=rbf, C=10, gamma=0.01 and class_weight=balanced[3] for SVM training hyperparameters as these were the hyperparameters which obtained the best score on the validation set.

4. Results

Our approach obtains a 75.0% accuracy and a 72.3% balanced accuracy on the test set. We have provided the score for the balanced accuracy because the label distribution is skewed towards SUPPORTED label. In Table 1 we have included this result as *BERT+SVM* combination. The baseline model for this task is the model that always returns the majority label which, in this case, is the label SUPPORTED.

Our results are not directly comparable to the ones obtained on the FEVER Shared Task as they have used a completely different setup, solving a similar (but different) task. The best accuracy obtained on their task is 68.2% obtained by the UNC-NLP group whose approach is explained in (Nie et al., 2018). That accuracy was computed on a FEVER Shared Task test set, not on ours, while solving a classification task with 3 labels and not our binary classification task. We also need to note that their test set had an equal distribution of labels, meaning that each of the 3 labels were represented with 1/3 of the test data. This label distribution means that their balanced accuracy and accuracy are the same. As we can not compare our result directly with theirs, the UNC-NLP FEVER results is not included in the Table 1 but still reported to gain some intuition in the difficulty of the task at hand.

Our BERT+SVM model beats the baseline accuracy by just 2% and often misses the label of a claim completely, as shown in Figure 5, where we can clearly see that there a couple of examples that should be REFUTED that have a near zero probability from the model to be REFUTED and the examples that should be SUPPORTED that have an over 0.8 probability to be REFUTED.

5. Conclusion

In this paper we took advantage of human-annotated claims created for FEVER dataset to create our own dataset for building a more general approach to fact checking. One that is not so dependent on Wikipedia and one that utilizes the entire web.

Unfortunately, we were not able to directly compare our results with any of the existing approaches as our task, and therefore dataset, is unique, at least to the best of our knowledge. Nonetheless, we argue that our approach obtains comparable results with the similar models.

In future works, it would be interesting to see comparison between different search engines (like Bing or Yahoo) and to find out which one of them would serve as a best fact checking engine.

[3]Balanced mode readjusts the error parameter inversely proportional to label frequency so that model can learn to classify into all labels no matter the label distribution. (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)

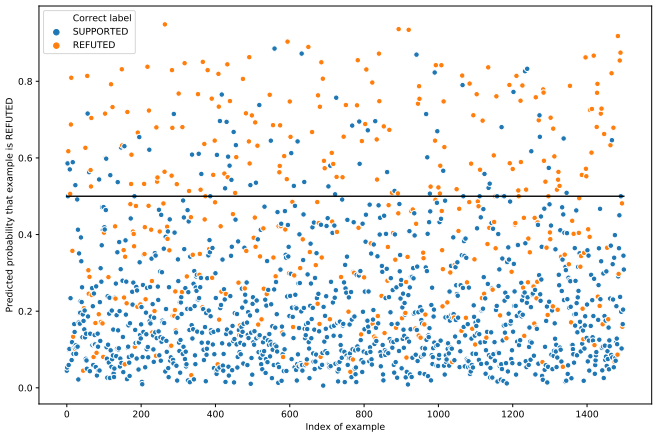


Figure 5: Predicted probability from the BERT+SVM model that an example from the test set is REFUTED. All examples above the black line are predicted as REFUTED while all under it are predicted as SUPPORTED. Color of the circle shows its true label.

6. Acknowledgements

We would like to thank Mladen Karan and Jan Šnajder for helping us decide the direction of our paper. Also we would like to thank Martin Tutek for helping us build and debug our Deep Learning model even though, in the end, we opted for a simpler model.

References

Tuhin Chakrabarty, Tariq Alhindi, and Smaranda Muresan. 2018. Robust document retrieval and individual evidence modeling for fact extraction and verification. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 127–131, Brussels, Belgium, November. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Yixin Nie, Haonan Chen, and Mohit Bansal. 2018. Combining fact extraction and verification with neural semantic matching networks. *CoRR*, abs/1811.07039.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018a. FEVER:

a large-scale dataset for fact extraction and verification. *CoRR*, abs/1803.05355.

James Thorne, Andreas Vlachos, Oana Cocarascu, Christos Christodoulopoulos, and Arpit Mittal. 2018b. The Fact Extraction and VERification (FEVER) shared task. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*.

Han Xiao. 2018. bert-as-service. https://github.com/hanxiao/bert-as-service.

Am I Just a Number to You? -Of Course Not! You're an N-dimensional Vector! How Different Fixed-size Representations Impact Clinical Text Classification

Lucija Šošić, Zorica Žitko, Ema Puljak

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{lucija.sosic,zorica.zitko,ema.puljak}@fer.hr

Abstract

The expert knowledge of a physician comes from years of studying different clinical texts and unique experience in working with patients. Rule-based systems have proven to be inefficient in capturing that knowledge and ever since researchers have been working on new methods to do that. The recent advances in both machine learning (ML) and natural language processing (NLP) have made it possible to process large amounts of data for successful inference, and healthcare is no exception - there are examples of ML-based systems exceeding even the performance of human experts. In this paper we explore different approaches to clinical text classification. Namely, we aim to determine how word and document representations impact the performance of both shallow and deep classifiers.

1. Introduction

Over the past decade genetic sequencing has received a lot of attention and significant breakthroughs took place. It is now possible to link certain genes to conditions and the other way around, giving rise to the idea of personalized medicine. Ideally, each patient would be treated differently, based on the unique combination of their genes and environmental factors affecting their condition. This would make the treatment much more effective and would furthermore lead to overall cost reductions in healthcare systems. One downside to this approach is that clinicians would be required to process huge amounts of information to come up with the optimal decision. Obviously, this calls for new methods of data management and processing, and machine learning seems to be particularly appropriate.

When dealing with tumors, scientists can detect any kind of mutation and larger genome alterations which make these diseased tissues dissimilar from the healthy ones. Some of the mutations, referred to as drivers, induce cancer growth, while others, called passengers, do nothing to contribute to cancer growth. Complexity of cancer genomes makes it challenging for the researchers to distinguish between the two types. Currently, classification of genetic mutations is being done manually by geneticists, pathologists and other experts, which is a very time-consuming, exhausting and consequently, error-prone task.

In this paper, we want to show how machine learning can be leveraged to classify clinical texts, more precisely, the ones regarding genetic mutations. Given the gene-mutation pair and clinical texts that refer to it, the goal is to determine how this mutation affects tumor growth and its malignancy. Having in mind that most of the provided texts are rather long (10,000 words on average), we explore different types of word and document fixed-size representations, that hopefully capture enough relevant information for successful classification, including tf-idf, Word2Vec and Doc2Vec. The representations are used as inputs to various shallow models and afterwards compared to more a recent, deep learning state-of-the-art model called BERT.

2. Related work

To the best of our knowledge, except for the 2018 Text Analysis and Retrieval Project Reports, there is not a single published paper concerning this particular dataset, therefore in this section we bring an overview of the work regarding clinical text classification in general.

In their work Pestian et al. (2007) tackle the task of assigning ICD-9-CM codes to radiology reports, resulting in the first freely distributable corpus of fully anonymized clinical text. Here, radiology reports mostly consist of free clinical text and the ICD-9-CM codes serve as a proxy to provide justification for instructed medical procedure. Furthermore, the authors emphasize how important well-structured data is for machine learning models.

Menger et al. (2018) aim to predict which patients will demonstrate violent behavior based on clinical texts from psychiatry. For this purpose, they evaluate several machine learning techniques, including the deep learning ones. This is similar to what we describe in following chapters, and yet different enough, due to inherently different natures of the underlying problems.

Obtaining high-quality numerical representations that capture even fine-grained relations between words and sentences is a well-known problem in NLP community. BERT (Bidirectional Encoder Representations from Transformers), developed by Devlin et al. (2018), is thought to be among state-of-the-art approaches on a specific set of tasks, which is why we intend to fine-tune it on our dataset in order to gain insight into its performance on texts from a quite specific domain and how it compares to shallow models. Furthermore, an extension of BERT to biomedical domain, BioBERT, was introduced by Lee et al. (2019) recently, but due to both time and computational limitations, will not be explored in the scope of this paper.

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 62–65, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

3. Dataset

3.1. An Overview

The dataset is provided by Memorial Sloan Kettering Cancer Center (MSKCC) that launched a Kaggle competition. It includes expert-annotated knowledge base of gene mutations. Both the training and test set consist of the clinical evidence used for classification and variants that provide the information about genetic mutation. Some of the test data has been machine-generated by the Kaggle platform to prevent hand labelling. There are nine different classes that have shown to be very unbalanced, as can be seen in Figure 1. Training set includes 3316 samples. There are 262 unique genes and 2993 unique variations. Maximum text length that can be found in the training set is 76708, whereas the shortest text is only 53 words long. The test set consists of 987 samples. It is obvious that the most challenging part of the task is to efficiently model clinical evidence (i.e. clinical texts), making the extensive comparison of different word and documents representations an important part of constructing a successful model.

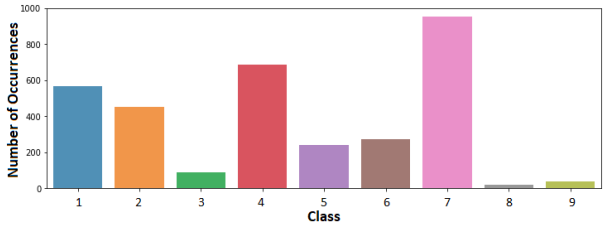


Figure 1: Class distribution on training data

3.2. Preprocessing

What precedes the representation of words using numerical vectors, are basic steps of text normalization that have proven to be crucial for transferring text from human language to a format that is suitable for a machine to read. Firstly, all text is converted to lowercase. Secondly, we applied tokenization, which was followed by stop words removal. "Stop words" are words that do not carry important meaning and make up a large majority of English words (e.g. the). Finally, the words were reduced to their stems (base or root form) through a process called stemming.

3.3. Different Representations

Before the training stage, four approaches to obtaining fixed-size words and documents representations were applied to both training and test data as follows. Firstly, we used the term frequency-inverse document frequency, known as tf-idf. This term-weighting scheme discovers which words are more important in a document and calculates the tf-idf value as a number of times a word has appeared in the document with the offset as the number of documents containing the word. The second set of representations is produced by Word2Vec, a model that captures different linguistic relations between the words. Each unique word in the corpus is assigned a corresponding vector in the vector space. As a result, the vectors that represent words found in similar contexts tend to be located in the nearby portion of the space. For generating these representations we used an open-source library fastText, created by Facebook's AI Research lab. Doc2Vec is a variation of Word2Vec that creates the representations of documents of different sizes. This algorithm tries to fix the weakness of traditional algorithms that do not consider order of the words and their semantics. While learning the vector representations of the words, it simultaneously learns the vector representation of the whole document. The fourth set of representations is a simple concatenation of Word2Vec and Doc2Vec representations. Even though word embeddings can be obtained from BERT (and BioBERT as well) after fine-tuning and could then be used to train some other deep learning model (e.g. LSTM), for the sake of simplicity, BERT is used as end-to-end solution for this task.

4. Models and Experimental Setup

4.1. Shallow Models

After having obtained different word and document representations as described in the previous section, we trained seven shallow models: logistic regression, support-vector machine (SVM) with RBF and linear kernels, decision tree, random forest, gradient boosting and voting classifier. The combination of classifiers for the voting classifier was determined by training a voting classifier for each combination of the previously mentioned models on training set and choosing the one with the highest micro-F1 score on test set for final evaluation.

Support-vector machine is an algorithm that determines where to draw the best decision boundary between vectors from different classes (or groups). To separate the two classes of data points, its aim is to find a plane that maximizes the distance between nearest data points of both classes. Linear and RBF kernel are different in terms of making the hyperplane decision boundary between the classes. They map the original data (linear/nonlinear) into a higher-dimensional space in order to make it linear.

Decision tree algorithm classifies inputs by segmenting the input space into regions. Each interior node corresponds to one of the input variables and leaf represents a value of the target variable. To classify a new input point, we simply traverse down the tree and at each node we ask a question about our data point.

Random forest algorithm is an ensemble of a set of classification trees where one of construction procedures is to randomly selected a subspace of features at each node to grow branches of decision trees. Afterwards, bagging method is used to generate training data subsets for building individual trees. Finally, all individual trees are combined to form random forest model (Breiman, 2001).

Gradient boosting produces a prediction from an ensemble of weak decision trees. It trains models in a way that next model corrects the errors of the previous one by using gradient in the loss function.

Voting classifier is a meta-classifier for combining either similar or conceptually different classifiers for classification via majority or plurality voting. Here, "hard" voting scheme is used, meaning that the final label will be the one most base classifiers assigned to the given instance, and the

chosen classifiers are decision tree, random forest, linear SVM and gradient boosting.

As the goal was to compare the impact of different word and document representations on models' performance (e.g. is it better to train linear SVM using tf-idf weighting scheme or Word2Vec), no hyperparameter optimization was performed, so there was no need to split training set to obtain the dev set.

4.2. BERT

BERT is a contextualized language representation model that uses "masked language model" to produce the representation. Since it covers both left and right context, it is used to pretrain Transformer, a multi-layer bidirectional encoder. It includes two separate mechanisms, an encoder that reads the text inputs and a decoder that produces a prediction for the task at hand. BERT expects the input data to be in the form shown on Figure 2. Furthermore, the data should be partitioned in a training, dev and test set, so 10% of training instances were selected at random to construct the dev set.

Not only does BERT expect input data in a certain format, but it also does not allow for the sequence length to be greater than 512 tokens. Obviously, this is far less than the average text length in our dataset, so we hypothesize that BERT's performance might be impaired compared to the cases when the whole input can be considered. To tackle the "short sequence" problem, we have used gensim summarizer to obtain the most relevant sentences from each text. Both summaries and original data were preprocessed as previously described and fed to BERT with maximum sequence length of 256 and batch size of 64. The model was trained for 8 epochs.

```
1   0   a  an example of text that should fit in class 0
2   1   a  an example of text that should fit in class 1
3   0   a  another class 0 example
4   2   a  a class 2 example
```

Figure 2: Formatting of the input as expected by BERT

We report the micro-F1 scores on the test set for all the models in the section that follows.

4.3. Results

For our baseline we have chosen linear SVM trained on Word2Vec representations on raw data (no preprocessing whatsoever). Table 1 shows micro-F1 score on test set for each classifier described in the previous section, the highest score for each model printed in boldface. Furthermore, we compare each of the models against the baseline (with micro-F1 score of 0.4125) via permutation testing at 90% significance level. Table 2 shows the corresponding p-values, where the smallest one in a row is boldfaced. Observing a p-value less than 0.1 allows us to reject the null hypothesis that micro-F1 scores obtained from the baseline and the model in question do not differ significantly, i.e if the model in question yields micro-F1 score greater than that of the baseline and p-value we get from permutation testing is less than 0.1, the observed gain is not simply due to chance or noise in the data. For each model permutation test was performed on 10000 permutations. As shown in Table 2, decision tree fails to outperform the baseline on this significance level, regardless of the representation. In contrast, all the other models are able to do so in most setups. More precisely, random forest trained with Doc2Vec, linear SVM and logistic regression trained with Word2Vec yield scores that are not significantly different from that of the baseline. Looking only at the results linear SVM trained with tf-idf representations yields without and with the described preprocessing (micro-F1 of 0.4125 and 0.6666, respectively, with p-value obtained through permutation testing of 0.047), we can tell that preprocessing has a rather positive impact on model's performance and certainly should not be disregarded.

	micro-F1 score			
	TF-IDF	W2V	D2V	WD2V
Decision tree	**0.5874**	0.5765	0.4672	0.5792
Random forest	0.6257	**0.6639**	0.5901	0.6366
Linear SVM	**0.6666**	0.5573	0.6284	0.6174
RBF SVM	0.2759	0.2759	**0.6448**	0.6257
Logistic regression	**0.653**	0.5765	0.6448	0.6066
Gradient boosting	**0.6612**	0.6284	0.6175	0.6338
Voting classifier	0.6612	0.6530	**0.6639**	0.6448

Table 1: micro-F1 scores on test set

	p-value			
	TF-IDF	W2V	D2V	WD2V
Decision tree	0.169	0.172	0.39	**0.153**
Random forest	0.094	**0.076**	0.112	0.095
Linear SVM	**0.047**	0.124	0.072	0.087
RBF SVM	0.091	0.098	**0.052**	0.060
Logistic regression	**0.044**	0.129	0.065	0.097
Gradient boosting	**0.054**	0.072	0.057	0.072
Voting classifier	0.063	0.075	0.053	**0.0489**

Table 2: p-values obtained through permutation testing on micro-F1 scores on test set

BERT's performance is observed separately because it does not use any of the mentioned representations, but rather creates them itself. Amazingly enough, its performance is comparable to that of shallow models, even though only a chunk of input text is used for prediction. The first setup where the model simply takes the first 256 tokens seems to work better than the one where the text was summarized (accuracy of 59,56% and 51,5%, respectively). This is possibly due to the fact that the most relevant information is present at the very beginning of the text, and yet, we find that hard to believe. Assuming the summarizer makes a reasonable amount of mistakes, if only the first few sentences were important, the results of summarization would still resemble the original texts, which they do not. Rather we hypothesize that as relevant as the chosen sentences might be, they fail to form a meaningful paragraph,

i.e. the context is ruined, which could negatively impact BERT's ability to produce high-quality contextualized representations of input sequences. This leads us to believe that connections between the sentences play an even more important role than we originally assumed. Furthermore, errors summarizer makes simply propagate when training BERT, which might be another reason why the results with the first 256 tokens are better than those with the 256 tokens obtained through summarization.

5. Future Work

The observed results do not seem good enough for any of the models to be used in clinical practice. Even though BERT has shown quite the potential despite using short sequences, we believe more could be done. Here is where we would start: BERT was pre-trained on a general corpus, whereas BioBERT was pre-trained on biomedical corpus, which almost certainly allows it to capture even more information from the input sequences in biomedical domain, such as those from this task, which is why we think this might just be the way to go. Furthermore, we would like to find out whether there is a more efficient way to filter out the important chunks of text or to somehow combine the outputs for each chunk to prevent data loss in cases like this, where the average text is simply too long.

6. Conclusion

The use of genetic information provides several benefits in aspect of personalized medicine. One of the biggest challenges is to extract information from biomedical literature in order to link genes and types of mutations that could lead to appearance of diseases.

We have shown how preprocessing using basic steps of text normalization of biomedical text greatly impacts the overall quality of obtained fixed-size representations. We provide extensive evaluation of different NLP approaches to forming word and document representations, including a contextualized language representation model. Surprisingly, such model, namely BERT, yields results that are not much worse than those of the shallow models, in spite of the limited input sequence length. We believe that the reported results could be further improved through using BioBERT and more sensible ways of contracting longer pieces of text to meet the mentioned limitations.

Acknowledgements

We would like to thank both Mladen Karan, PhD and associate professor Jan Šnajder, PhD for much needed useful advice and patience to answer our numerous questions.

References

Leo Breiman. 2001. Random forests. *Machine learning*, 45(1):5–32.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. Biobert: pre-trained biomedical language representation model for biomedical text mining. *arXiv preprint arXiv:1901.08746*.

Vincent Menger, Floor Scheepers, and Marco Spruit. 2018. Comparing deep learning and classical machine learning approaches for predicting inpatient violence incidents from clinical text. *Applied Sciences*, 8(6):981.

John P Pestian, Christopher Brew, Pawe—ł Matykiewicz, Dj J Hovermale, Neil Johnson, K Bretonnel Cohen, and W—łodzis—ław Duch. 2007. *A shared task involving multi-label classification of clinical free text*.

Early Depression Detection Using Temporal and Sentiment Features

Donik Vršnak, Mate Paulinović, Vjeko Kužina

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{donik.vrsnak, mate.paulinovic, vjeko.kuzina}@fer.hr

Abstract

Major depressive disorder is one of today's most pressing problems. This makes early depression detection especially important. Studies have shown that a person's use of language can be indicative of their mental state. Furthermore, the vast repository of text generated by users on social media platforms provides valuable resources for research. In this paper, we approach the task of depression detection through different machine learning models and feature engineering, focusing especially on temporal features and user sentiment. We used a dataset comprising of post histories of Reddit users. We present the results in this paper and show the effect of temporal and sentiment data on classification.

1. Introduction

Major depressive disorder (MDD) is a common mental disorder that is associated with normal emotions such as sadness that do not remit when the external cause of those emotions ceases (Belmaker and Agam, 2008). One of the major problems with diagnosing depression is that it requires a distinct change in mood that lasts at least two weeks and has considerable effect on the psychophysiological state, such as disturbances in sleep or appetite. Because of this, diagnosis usually relies on self reported symptoms and experiences of the patient. However, many people who show signs of depression are not aware of it. This is why it could be useful to create a system for early detection which could alert the patient that they might be suffering from some form of depression.

With the rapid growth of social media, there has been an explosion of user generated data which could offer some insight into their emotional state. The link between the language use and the emotional state of a person has long been a subject of research, and has shown that there might be a significant correlation between the two (Rude et al., 2004). Because of this, in recent years, there has been an effort to construct a machine learning model that could predict whether someone suffers from MDD, usually based on that person's social media presence. The idea behind these models is that changes in someone's social media activity might indicate a change in the mental state of that person, which could be a result of depression.

In this paper, we propose a simple model for the detection of possible depression characteristics from social media posts and comments. The model classifies a person into one of two classes based on whether they show signs of depression in their post history or not. The data used to train the model is based on data composed in Losada and Crestani (2016), and is composed of Reddit posts in a period of three years. We also ask the question about the usefulness of features that take into account coarse grained temporal data on relatively small number and relatively high average length of posts, and whether this type of features provides any new information when used in conjuncture with text based features.

In the first part of the paper we focus on features and feature selection, especially on feature engineering that was done in preparation for model training, and in the second part we explain our models, pick the best model and present the results. We also comment on what are some possible reasons for some features failing to provide new information. The model is publicly available.

2. Related Work

A lot of psychiatric research has been done in relation to depression, specifically around the possible causes and symptoms of depression. Also, there have been many studies connecting linguistic analysis of written and spoken text to predicting and diagnosing depression (Rude et al., 2004). In recent years there have been several papers about depression detection, as early detection could prevent loss of life. One of the first attempts to predict depression from social media was proposed by De Choudhury et al. (2013b), where they use tweets paired with anonymous user diagnoses, which they acquired using crowdsourcing, as their training data. They represent an user by their posts, mentions contained in those posts and they track those features through time. They do this by fitting a curve through the data, so that the feature encodes only the parameters of that curve. On the other hand Losada and Crestani (2016) focused more on creating a large solid dataset from data retrieved from Reddit. They also proposed a novel way of including temporal data, which does not require feature engineering, but instead uses a custom loss function that penalizes the model based on how quickly it is able to detect depression characteristics.

In our work, we propose temporal features similar to those introduced in (De Choudhury et al., 2013b), but we use them to model sentiment changes through time. Also we use another type of temporal features that do not rely on curve fitting, but instead simply represent the data in some fixed points in time (for example, one such feature is the total number of posts in every hour of the day). We combine these features with other features retrieved directly from the user's posts and comments to create a feature representation for that user.

Text Analysis and Retrieval 2019: Course Project Reports (TAR 2019), pages 66–70, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2019.

Table 1: Details of the dataset used for training and testing

Dataset	Users	Posts	Comments
Positive Train	83	4911	25940
Negative Train	389	81944	158864
Positive Test	54	1929	16800
Negative Test	352	65754	151947

3. Dataset

The data used in this paper was created by Losada and Crestani (2016), and it consists of samples marked as either depressed or not depressed. The samples consist of the post and comment history from users of the social media platform Reddit, where each of their comments and posts in the last few years comes with a timestamp. The positively marked users were ones which gave a clear and explicit mention of a depression diagnosis and were manually reviewed afterwards. The negatively marked users were selected either fully randomly or ones which were active on a sub-forum dedicated to depression but were not depressed, so that the collection becomes more realistic.

The data is split into train and test categories, with train being slightly larger at 472 users, of which 83 were positive (i.e. depressed) and 389 were negative. The test set contains 406 users, 54 of them in the positive class and 352 in the negative class. Because there exists a large disbalance of classes, we tried both oversampling and undersampling methods, but after testing we settled on using simple random oversampling of the positive class for our train set. Dataset statistics is shown in in Table 1.

4. Features

For our classification task, we set out to create features that could encapsulate users behavior over a given time period. The idea for these features came from the paper (De Choudhury et al., 2013b). We reason that these features could provide useful information because of the definition of MDD as an illness that must last longer than two weeks. That is why we propose a feature that describes the trend of any sort of numerical data in time. We refer to this feature as *TimeTrendFeature* and it provides 6 different measures and can be represented as a 6 dimensional vector:

1. **Mean** - average of data points across the whole time period
$$\frac{1}{N}\sum_{t=1}^{N} X_t \tag{1}$$

2. **Variance** - the variance of data points across the whole time period
$$\frac{1}{N}\sum_{t=1}^{N}(X_t-\mu)^2 \tag{2}$$
where μ is the mean over all data points

3. **Mean momentum** - We define momentum as the trend of the data points of one part of the time period in relation to previous M parts of the time period.
$$\frac{1}{N}\sum_{t=2}^{N}\left(\frac{X_t-1}{t-M}\sum_{k=M}^{t-1}X_k\right) \tag{3}$$
In our case, whole time period was separated into months and we looked at only the previous month $(M = 1)$. We decided to use month based features to add a sort of buffer period around the 2 weeks specified in the definition of MDD.

4. **Entropy** - the measure of uncertainty in data points
$$-\sum_{t=1}^{N} X_t \log X_t \tag{4}$$

5. **Max** - largest data point
$$\max_t X_t \tag{5}$$

6. **Min** - smallest data point
$$\min_t X_t \tag{6}$$

We use this type of feature to try to model two things:

- How the sentiment of messages changes over time
- User activity over time

For sentiment, we measure average positive and negative sentiment and their variance over all posts in a month, and then use trend features to represent them over the whole post history. Sentiment analysis was done using VADER sentiment (Hutto and Gilbert, 2014), a lexicon based tool used to determine sentiment of words and sentences.

Similarly we calculate the average and variance of post count through the whole post history. The main reason we model the activity is based on the assumption that a significant change in activity could be indicative of a change in users social habits, which could be linked to depression.

We also propose some additional time-agnostic features:

- tf-idf scheme based on 1-grams and 2-grams that are used to model users' post history (constructed over the whole post history for every user)
- BOW scheme that is also based on 1-grams and 2-grams (constructed over the whole post history for every user)
- Users' posting habits throughout the day by counting the number of posts done at different hours (represented as a 24 dimensional vector)
- Ratio of user comments versus posts in the whole post history

On tf-idf and BOW vectors, we try to use a reduction function based on sentiment, which removes all n-grams that have neutral sentiment, but it did not have a strong impact on model performance. We use the before described trend features to augment these features. The final feature vector used by our model is constructed by simply concatenating all of the feature vectors.

Furthermore we extract the number of emojis a person uses as well as the number of exclamations and pronouns such as "me", "myself" and "I", and add them as possible features to our vector. All of them have the same motivation which is that a depressed person would use them more then a non depressed person (Rude et al., 2004).

5. Models

We focus on two types of models for the classification task: a voting ensemble and a neural network. We tried various combinations of different features in search of the best combination of model and features. We compare both of our types of models to each other and to a simple baseline. We present the results of these comparisons in Section 6.

5.1. Voting Ensemble

After preliminary testing of machine learning models the best results were obtained using a combination of logistic regression, random forests and a ridge regression classifier models, while support vector machine, AdaBoost, and Naive Bayes classification models did not show promising results. The final decision for the model fell on a voting classifier which took into account the votes of the logistic regression, random forests and ridge regression classifier in hopes of increasing the performance through majority voting.

5.2. Neural Network

Following some preliminary testing, we decided to use a simple feedforward neural network model consisting of 3 hidden layers with ReLU activation function. The input into the network was a feature vector representing the whole post history for the user and the output is defined by a single neuron with a sigmoid activation function. For training the network we use both SGD and Adagrad optimization method with a goal of minimizing binary cross entropy loss function.

6. Results

6.1. Experiments

We conducted many experiments to try to determine whether these kind of feature engineering could provide useful information for depression detection. We performed these experiments independently for both the voting classifier and the neural network to try to come up with the best possible set of feature for each model. After each round of feature selection we performed cross validation to determine the optimal parameters. For scoring the models we used F1 score on the positive class, as accuracy is not representative because of the class imbalance. For both the ensemble and the neural network the first set of tests was conducted only on vectors obtained directly by vectorizing text. These included BOW, tf-idf and the polarity reduced BOW vectors. We determine that tf-idf vectors performed the best. Our final ensemble and neural network results are shown in Table 2. The best result was obtained by our ensemble trained on the tf-idf features and the difference to the baseline was shown to be statistically significant with a p-value of 0.05. Further information regarding the model parameters are in the following sections. The second set of features we tested included every engineered feature concatenated to same vectors as in the previous set. For brevity, we show the results for only the best experiments.

6.2. Voting Ensemble

For the experiments on the voting ensemble nested cross-validation was conducted on the random forest, logistic and ridge classifier independently for both test sets, to find out the best hyperparameters for each of them, after which they were combined into a voting classifier to hopefully upgrade their performance. The result was numerically greater in the first set as shown in Table 2, but the difference was not shown to be statistically significant.

6.2.1. Neural Network

The experiments used for feature selection for the neural network can be separated into 3 sets. Detailed results for each set can be found in Table 2. In the third and final set we tried to manually select a subset of features that would improve the scores of the network. After extensive testing we concluded that by using sentiment trend features and the ratio of comments in the whole post history, in combination with 200 dimensional tf-idf vector and the count of exclamation characters in posts, we got the best results from the neural network model.

6.3. Discussion

Table 3 shows our best classifier in comparison to the best classifiers in (Leiva and Freire, 2017) and (Losada et al., 2017). We see an numerical improvement of the F1 score by a small margin over the previous best result (0.64), although the significance of it is questionable.

From the results we can conclude that our trend features do not seem to add any new information which the models can learn about the user. We believe that this happens because of two main factors, both related to granularity. Firstly, grouping user data into months could be too coarse-grained, and because of that finer trends could be lost in the noise, especially since we only store the mean momentum for each user. Also, another reason could be that, even though these features seem to work well on smaller, more numerous texts (such as tweets) (De Choudhury et al., 2013b), (Coppersmith et al., 2014), they could be too coarse-grained to be able to accurately represent subtler changes in sentiments in longer form posts, such as Reddit posts used in this paper.

Furthermore, based on observations made in the paper by De Choudhury et al. (2013b) about tweeting time being significantly different between classes, we tried to incorporate the same feature based on post times. However, after

Table 2: Experiment results

Model	F1 score
Random classifier	0.21
Neural network without posting time features	0.38
Neural network with all features	0.38
Neural network without tf-idf features	0.41
Neural network without reduced polarity and posting time features	0.42
Ensemble with tf-idf features, temporal and text features	0.62
Ensemble with tf-idf features	**0.65**

Table 3: Experiment result comparison with (Leiva and Freire, 2017) and (Losada et al., 2017)

Model	F1 score
Genetic algorithm and PCA	0.59
Ensemble	0.60
FHDOA	0.64
Ensemble with tf-idf features	**0.65**

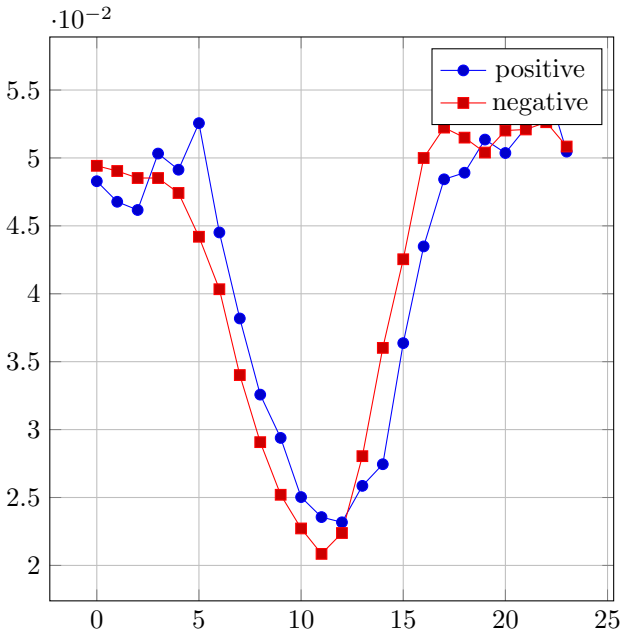


Figure 1: Shows that posting times tend to be the same across both classes

testing, we concluded that this feature is not suitable for Reddit posts. More details are shown in Figure 1. It appears that posting times tend to be fairly similar for both classes. However, it is interesting to note that the curves shown in the figure closely resemble those of the depressed class acquired from tweets (De Choudhury et al., 2013b).

For the suggested emoji and 1st person pronouns it was not shown that they add any significant information, which we attribute to the idea that the tf-idf representation already takes that information into account if it is useful.

7. Future Work

For future work it would probably be beneficial to compile a dataset with more fine-grained information about the activity times of the users, and somehow rule out possibilities of depressed people being labeled as not depressed in the dataset. The time frame used by trend features could also be reduced from a month to the lower bound of two weeks needed for depression detection based solely on the medical definition of depression (Belmaker and Agam, 2008).

8. Conclusion

In this paper, we set out to create new types of features that could be used to classify data from social media posts into depressed and non depressed categories. To this end, we proposed a set of time based features that could, in theory, encapsulate trends that appear in the data based on the work by De Choudhury et al. (2013a). We have trained two types of models, a voting classifier and a simple feedforward neural network. We show that our voting classifier outperforms the network model. While training our models, we have used different sets of features, and we have shown that our temporal features do not seem to add any new information that could significantly increase the F1 score of the model. Furthermore, time needed to preprocess the data rose significantly when we used these features, as significant computational power is needed to compute some of them, especially for sentiment based time features.

References

RH Belmaker and Galila Agam. 2008. Major depressive disorder. *New England Journal of Medicine*, 358(1):55–68.

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.

Munmun De Choudhury, Scott Counts, and Eric Horvitz. 2013a. Social media as a measurement tool of depression in populations. In *Proceedings of the 5th Annual ACM Web Science Conference*, pages 47–56. ACM.

Munmun De Choudhury, Michael Gamon, Scott Counts, and Eric Horvitz. 2013b. Predicting depression via social media. In *Seventh international AAAI conference on weblogs and social media*.

Clayton J Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*.

Victor Leiva and Ana Freire. 2017. Towards suicide prevention: Early detection of depression on social media. In *International Conference on Internet Science*, pages 428–436. Springer.

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. erisk 2017: Clef lab on early risk prediction on the internet: experimental foundations. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 346–360. Springer.

Stephanie Rude, Eva-Maria Gortner, and James Pennebaker. 2004. Language use of depressed and depression-vulnerable college students. *Cognition & Emotion*, 18(8):1121–1133.

Author Index