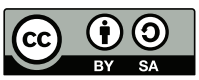
Text Analysis and Retrieval 2017

Course Project Reports

University of Zagreb

Faculty of Electrical Engineering and Computing



Publisher:

University of Zagreb, Faculty of Electrical Engineering and Computing

Organizers:

Domagoj Alagić
Mladen Karan
Jan Šnajder

Editors:

Domagoj Alagić
Mladen Karan
Jan Šnajder

ISBN 978-953-184-237-2

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 fourth booklet in a series of students' project reports from the Text Analysis and Retrieval (TAR) course taught at FER, University of Zagreb. TAR teaches the foundations of natural language processing and information retrieval methods, as well as their applications and best practices through hands-on practical assignments. The relevance and importance of these skills in today's information-saturated world hardly needs any elaboration. Many doors open to those who acquire the skills of text analysis in particular, and data analysis and machine learning in general. We are therefore happy to be able to help open these doors for our students.

TAR 2017 Projects Report booklet presents the results of 14 projects, which are the work of 37 students. The topics – most of which adopted from the recent SemEval, CLEF, and TREC shared tasks – include author profiling (5 papers), Twitter sentiment analysis (3 papers), semantic textual similarity (2), temporal text analysis (1 paper), humor prediction (1 paper), text simplification (1 paper), and factoid question answering (1 paper). Regarding the methods, most teams have relied on traditional machine learning algorithms (SVM being the clear winner), while only two teams dared to give deep learning models a try.

All project reports are written in the form of a research paper. This, of course, is deliberate. In our view, teaching the students to present their results effectively is as important as teaching them the actual methods and tools. Moreover, this time around students also participated in paper reading sessions followed by in-class discussions. Curiously enough, for many of them, this was the very first time that they were asked to read scientific papers. The students' feedback was overwhelmingly positive, and we wouldn't be surprised if reading sessions have helped in improving the quality of some of the papers in this booklet.

As the course organizers, we thank the students of TAR 2017 for the motivation and enthusiasm they demonstrated. For us, TAR 2017 has been a truly rewarding experience.

Domagoj Alagić, Mladen Karan, and Jan Šnajder

Contents

SkiBaVotE: Modelling Skill-Based Voting for Complex Word Identification

Antonio Alić, Tomislav Lokotar, Leonard Volarić Horvat

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{antonio.alic,tomislav.lokotar,leonard.volaric-horvat}@fer.hr

Abstract

Complex Word Identification is a part of a much broader problem called Lexical Simplification. Its goal is to try and detect certain words which might prove challenging to certain readers, like children or non-native speakers. This paper describes a simple and intuitive method of using a non-probabilistic SVM voter ensemble, which incorporates a voter skill prediction method, to tackle the problem of CWI. Voter skill is modelled by assigning incremental weights to certain voters for word complexity, and decremental weights for word simplicity. Even though our system, SkiBaVotE, was merely exploring the idea, it performed quite well. Therefore, further work in this direction might prove fruitful.

1. Introduction

This paper will cover the problem of Complex Word Identification (CWI), which is a part of the larger Lexical Simplification (LS) pipeline. Lexical Simplification is the process of analyzing a given text, finding words and phrases which are deemed complex for a target audience, and substituting them with simpler, semantically and grammatically roughly equivalent alternatives. There are a lot of people who might benefit from LS: such a system could make a text more accessible not only to a specific target audience, but also to, for instance, non-native speakers, readers of different ages and education levels, laymen, mentally challenged readers etc. The whole LS process can be broken down into a pipeline, with CWI at its beginning, substitution selection (e.g. *feline* $\rightarrow$ *cat*) in the middle, and a (hopefully) simplified sentence at its end. Since Complex Word Identification is a crucial part of LS, CWI-related work, approaches and problems will be laid out in this paper.

One of the most important tasks we must perform is feature extraction. We have chosen a wide variety of features, broadly divided into three types: boolean features (e.g., the presence of a word in a given medical dictionary), integer features (e.g., word length or vowel count) and float features (e.g., average letter weight in a word). Such features, along with *word2vec*, will be the basis for training our Machine Learning model.

Our system, dubbed SkiBaVote (**Ski**ll-**Ba**sed **Vot**er **E**nsemble), consists of an ensemble of eight SVM voters with inversely weighted outputs: "complex" output weights increase, while "not complex" output weights decrease, in an attempt to model English language skill of eight speakers: novice speakers with the first voters, intermediate speakers with the middle voters, and excellent speakers with the last voters. This is explained in depth later in the paper.

Finally, we will discuss the results of our attempts at tackling the CWI problem using the aforementioned system.

2. Related Work

Since CWI is a key part of Lexical Simplification, most related work pertains primarily to LS, while CWI implementation is merely mentioned or explained very briefly. One of the rare papers dedicated to CWI itself (Shardlow, 2013) brought forward three approaches to tackling the problem. Since it was written comparatively early, it includes fairly naïve approaches. The approaches are:

- Simplify Everything – a bruteforce method, simplifying every relevant word. It is somewhat useful as a baseline system, and minor tweaks, like limiting the approach to simplifying only nouns, adjectives and verbs, can help a bit.
- Frequency Thresholding – the metric used to determine whether a word is complex or not is its frequency on the level of the whole text or corpus. This approach is more interesting than the last one, but it is based on only one feature. SkiBaVotE uses many metrics of a similar nature, as seen in the rest of the paper.
- Support Vector Machine – an efficient machine learning model, taking various features into account. This concrete SVM was fed only six features, like frequency or syllable count. Its results leave much to be desired: the precision is slightly higher, but the recall is drastically diminished. Our system also utilizes SVM, but we gave it more than 300 features (described later). The reasoning behind this is that six features are simply not enough, and that many more features might improve the results.

In (Paetzold, 2015), each segment of the LS pipeline is described separately. This, naturally, includes CWI, but puts it in context with LS and provides some "big picture" details, so that work was by far the most useful to us. The work mentions several CWI strategies, most prominent of which are Lexicon-Based, Threshold-Based and Machine Learning-Assisted. Due to the radically different, and yet very promising concepts of each, we decided to use these together.

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 1–5, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

Table 1: Example dataset entry.

Original Sentence	*The new distillery owner, Samuel Vimes, employed many people.*
Tokenized Sentence	*The new distillery owner , Samuel Vimes , employed many people .*
"distillery" evaluation	*The new distillery owner , Samuel Vimes , employed many people . distillery 2 1 0 1 0 0 0 1 0*
"people" evaluation	*The new distillery owner , Samuel Vimes , employed many people . people 10 0 0 0 0 0 0 0 0*

3. Dataset

3.1. Description

Complex Word Identification was also a task problem during the 2016 SemEval (**Sem**antic **Eval**uation) competition, so we used the dataset provided for the competition.[1] It consisted of the training set (2,237 instances) and the test set (88,221 instances), and was formed as follows:

- 400 annotators were given various 20-to-40-word long sentences to determine which ones they did not understand.
- Each of the 200 sentences in the training set was annotated by 20 distinct annotators.
- The test set was annotated over 9000 sentences by only one (distinct) annotator.
- A word is deemed complex if at least **one** annotator named it complex.

Upon closer inspection, we determined that the training set contained certain peculiar entries. For example, contraction parts (like "'ve" or "n't") were put forth for evaluation, and perhaps the most bizarre anomaly is that one annotator annotated the verb "do" – perhaps the most common auxiliary verb in the English language – as a complex word. [2]
Regarding the distribution of complex and simple, the training set has 30% complex words whereas the test set has only 5% complex words. This, along with the vast difference in training/test set sizes, might be responsible for the relatively modest success of most systems submitted during that SemEval, as well as our system.

A single training set entry consists of three parts: the whole sentence (after tokenization), a token being classified (along with its zero-based index in the sentence), and annotator outputs (a list of 20 zeros or ones).

3.2. Entry Example

An example sentence would be:
The distillery owner, Samuel Vimes, employed many people.
After being put through a tokenizer, each candidate word is evaluated separately. Note that the interpunction is tokenized and indexed as well. Also, even though it was not explicitly stated, the dataset seems to have been formed with the help of a Named Entity Recognizer, as no proper nouns came up as candidate words anywhere in the whole dataset. Two sample entries, evaluating the words "distillery" and "people", are shown in Table 1.

Three annotators rated "distillery" as complex, so the word should be classified as complex. On the other hand, not a single annotator called the word "people" complex, so we can conclude that it is not complex, and should not be classified as such.[3]

The test set format is largely the same, the only differences being in size and in the fact that each word has only one annotation, instead of the 20 used in the training set.

4. Feature Extraction

4.1. CWI Approach Analysis

As mentioned before, Paetzold (2015) proposes several CWI strategies, as briefly described ahead:

Lexicon-Based. Lexicon-based strategies make use of predetermined dictionaries and lexicons. The idea is that, given a lexicon that contains, for instance, simple words, a word could be classified as simple (if it is in the lexicon) or complex (if it is not in the lexicon). The concept is simple, and lexicons can be easily obtained, so this can be used as a rudimentary, "quick and dirty" classifier in itself. Several online resources proved to be very useful for this task – namely the Simple Wikipedia.[4]

Threshold-Based. Threshold-based strategies are a tad more challenging, but (generally) do not depend on the availability of a lexicon. Instead, we take a metric of a word – like word length, syllable count or corpus-wide word frequency – and set a threshold value. Then we, naturally, classify the word based on whether the metric crosses the threshold or not. For instance, the word "chair" is very common – meaning its frequency is high (higher than a threshold we set beforehand) – so it probably will not be percieved as complex. The word "paraplegics", on the other hand, is fairly rare – its frequency is below the threshold – so it could become a likely candidate for being evaluated as complex.

[1] `http://alt.qcri.org/semeval2016/task11/`

[2] We concluded that it might be interesting if we raised the threshold for the number of annotators required to mark a word as "complex", in order to avoid such anomalies. The results of this, however, were not as good as we had hoped – in fact, they were worse – so we dropped the idea.

[3] The example is not from the dataset – it was made up to have all relevant fields, but the sentence is much shorter than those from the actual dataset, and only eight annotators were simulated instead of the usual 20, for the sake of brevity.

[4] `https://simple.wikipedia.org`

Table 2: The complete list of features used in our model.

Feature Name	Feature Explanation
Weighted Letters	Weights letters of the word by their average frequency in the language
Is Capitalized	Tracks whether a word is capitalized or not
Is Acronym	Tracks whether a word is an acronym (all capitalized)
Contains Contractions	Tracks whether a **sentence** contains contractions like "can't" or "doesn't"
Stem Distance	Determines the Levenshtein distance between the word and its stem
Is English	Tracks whether a word is in the used English dictionary
Is Medical Term	Tracks whether a word is in the used medical dictionary
Word Length	Determines the word length
word2vec	A *word2vec* vector containing 300 features
PoS Tag	One-hot encoded Part-of-Speech tag of the word
Consonant Count	Determines the number of consonants in the word
Consonant Pair Count	Determines the number of consonant bigrams in the word
Consonant Triple Count	Determines the number of consonant trigrams in the word
Sentence Length	Determines the number of words in the sentence
Average Word Length	Determines the average word length of a sentence
Weighted Average Word Length	Weighs the words of the sentence by the average word length
Word in Top 20k	Tracks whether a word is in the most frequent 20k English words
Simple Wikipedia Frequency	Determines the word's Simple Wikipedia occurrence frequency

Machine Learning Assisted. Machine Learning (ML) models are very interesting for CWI because they are far more flexible than the rule-based systems listed just above, and should handle the problems of natural language (like ambiguity or vagueness) better than rule-based systems. An ML-assisted approach to CWI is generally not in focus in related CWI (or, rather, LS) work – for instance, (G. Glavaš, 2015) has no notion of ML, and Shardlow (2013) tried implementing an SVM and concluded that the results were not good, but his approach was, as mentioned, perhaps a bit too naïve.

On the other hand, (L. Specia, 2012) presents the findings of the first English LS SemEval, where it can be seen that the best submitted systems implemented an SVM approach.

4.2. Feature List

Since SkiBaVotE makes extensive use of ML models – namely, SVM, described in detail later – it is imperative that we make an extensive list of hopefully relevant features of words. The features observed in our work were largely chosen with regard to the aforementioned CWI strategies laid out in (Paetzold, 2015) and described above.

Unlike Shardlow (2013), who used an SVM with only six word features, we decided to give our SVM over 300 features, the majority of which come from *word2vec*[5]. Most are context-independent, while some are context-oriented. The full list of features is given in Table 2, along with a short description of each.

[5]Since Shardlow's paper was published within a month after *word2vec* was introduced, it is possible that, at the time of researching, he simply did not have *word2vec* or similar pretrained word embeddings at his disposal. Still, even without *word2vec*, our model has a lot more features than his.

We originally used *word2vec* for the word before and the word after the current word, but dismissed it because we determined it was redundant, based on performance results.

5. SkiBaVotE System Description

As mentioned before, SkiBaVotE is based on SVMs, through the use of an ML voter ensemble.

Baseline. For the baseline, we used a single SVM, using the features listed in Table 2. The decision to use SVMs was made because we thought that Shardlow (2013) had a good approach, but could have benefited from using more features, which our SVM clearly uses.

SVM Voter Ensemble. Our SVM ensemble consists of 20 SVM classifiers which play the role of voters. These 20 classifiers have variable weights.

Hyperparameters. Each voter's hyperparameters were determined individually, by performing a grid search over a wide array of hyperparameters, using 5-fold cross-validation on the training set.

5.1. Weighting Scheme.

The weighting scheme is inverse for the two outputs: the first voter has the highest weight for the "not complex" output and the lowest weight for the "complex" output. The last voter has the exact opposite weighting scheme. The weights range from one to five, and change linearly. In order to illustrate this, it will be written in the form of a matrix:

$$\begin{bmatrix} 5 & 4.95 & 4.9 & \dots & 1.1 & 1.05 & 1 \\ 1 & 1.05 & 1.1 & \dots & 4.9 & 4.95 & 5 \end{bmatrix}$$

Each column represents a voter. The top row represents the weights of the voters' decisions for the output "not

complex", and the bottom row represents the weights for the output "complex", so that if the last voter rates a word as complex, then it is highly probable that it indeed is complex, whereas it does not matter as much if it rates a word as simple. The exact opposite line of reasoning can be applied to the first voter – and, of course, all of the voters in between.

5.2. Voting Scheme

Each of the voters is trained on the number of word annotations from the training set. So, if there is one "complex" annotation, the first voter will fire, and if there are, for instance, 5 annotations, the first five voters will fire etc.

After training the model, we concluded that the word was practically guaranteed to be rated "complex" for a certain number of annotations, and further annotations simply conveyed little-to-no new information. Therefore, after careful deliberation, we decided to remove the surplus SVMs, and went from the original 20 voters down to the eight essential voters.

5.3. Motivation for the Voting Scheme

The idea is that the lower voters (dubbed "newbie") model novice and intermediate speakers, while the higher voters (dubbed "pro") model good and excellent speakers.

The motivation behind the idea is this: if the members of the *newbie* group classify a word as simple, then it has a high probability of indeed being simple, whereas their votes carry less weight when deciding if a word is complex. The analogous line of reasoning is used for the *pro* group – their vote carries more weight when classifying words as complex, and less weight when classifying words as simple.

At first glance this might seem like we are neglecting the votes of the *newbie* group – which should be the target audience for CWI – but this was shown to work very well. The main point here is that – perhaps counterintuitively – we do not want the LS system to simplify a given text *too much*, because minor semantical and grammatical differences (inherent to practically each substitution) tend to add up and potentially change the given text too drastically.

6. Evaluation and Discussion of Results

After evaluating it on the test set, we concluded that SkiBaVotE outperforms our baseline. The difference is statistically significant with $p<0.05$. Table 3 shows precision, recall, F1 and G1[6] scores of five systems: SkiBaVotE, our baseline, and top three systems (F1-wise) from that year's SemEval CWI task. It should be pointed out that SemEval competitors were required to optimize their results for G1-score, whereas we optimized for F1-score. The unusual nature of the dataset, as stated in Section 3., is also responsible for the comparatively low scores.

Table 3: Comparison of SkiBaVotE to selected CWI systems from other authors.

System	Precision	Recall	F1	G1
SkiBaVotE	**0.371**	0.520	**0.433**	0.668
Our baseline	0.193	**0.608**	0.293	**0.713**
PLUJAGH - SEWDFF	0.289	0.453	0.353	0.608
LTG – System1	0.220	0.541	0.312	0.672
LTG – System2	0.300	0.321	0.310	0.478

It is noteworthy that our baseline's recall was higher than SkiBaVotE's. This is, however, perfectly acceptable, because too big a recall is not desirable when doing LS, as explained in Subsubsection 5.3..

7. Problems and Future Work

While working on this paper, we encountered problems primarily pertaining to the nature of the dataset. Firstly, the distribution of the training set (30% of the words being complex) was vastly different than that of the test set (only 5% of the words being complex). Secondly, it contained many anomalies and highly unusual situations, such as the word "do" being annotated as complex, or contraction parts, like *'ve* or *n't*, being evaluated at all. Finally, it was unclear from the dataset description why there were no named entities or capitalized words being evaluated, both of which are easily imaginable as real-world examples.

Our weighted approach to aggregating annotator information performed well, but it would be interesting to see if that information could be aggregated even better. Also, there might be a better way to accomplish our weighted approach, perhaps by implementing a probabilistic model instead of our linear function.

Furthermore, better results might be achieved by combining our model with an unsupervised approach, as well as by boosting confidence in a scenario where there is an equal number of voters for each output.

Finally, it should be pointed out that, if we observe CWI within the context of LS, a very high recall is, in this specific case, not something to be desired, due to the fact that, if too many words are classified as complex and put forth for substitution, then all the minor semantic differences from substitutions will add up and change the original text's semantics potentially too much for it to be acceptable.

8. Conclusion

In this paper, we presented our approach at performing Complex Word Identification. Previous CWI-related work has shown that systems using various word and sentence features, combined with an adequate machine learning model, show the most promise. To that end, we decided to use an SVM ensemble, and selected many features –

[6]F1-score is the harmonic mean of precision and recall, whereas G1-score is the harmonic mean of accuracy and recall.

lexicon- and threshold-based, as well as the ones generated by *word2vec* – to try and get meaningful results.

The dataset used was provided by the organizers of the 2016 SemEval competition, which featured a CWI task as well. That dataset proved to be somewhat challenging, as it was not made clear how or why certain elements of it were formed. Most of our features are generic word or sentence features, not specific to the dataset itself, but the dataset contained annotations which we were able to use for training our model. The way this was done is by basing the SVM ensemble on several voters, and then assigning variable decision-making weights to them, in an attempt to model their language proficiency. This was proven to work surpringly well, but it is evident that there is a lot of room for improvement – for instance, by distributing the voter weights in a better fashion, or combining our approach with an unsupervised model.

References

S. Štajner G. Glavaš. 2015. Simplifying lexical simplification: Do we need simplified corpora. In *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics*.

R. Mihalcea L. Specia, S. K. Jauhar. 2012. English lexical simplification. In *First Joint Conference on Lexical and Computational Semantics*.

G. H. Paetzold. 2015. Reliable lexical simplification for non-native speakers. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

M. Shardlow. 2013. A comparison of techniques to automatically identify complex words. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*.

Stereotypes Can Be in Writing Too - Age and Gender Identification of Twitter Users

Ana Brassard, Tin Kuculo

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ana.brassard, tin.kuculo}@fer.hr

Abstract

In this paper, we observe and compare the performances of various machine learning algorithms for the task of identifying the age group and gender of tweet authors. We follow previous work on this topic from the PAN@CLEF 2016 competition and attempt to push forward the results using a similar base approach to one of the contestants, which is representing the tweets as a tf-idf weighted matrix. Along with the data used in the competition, we did some additional testing with a separate dataset of tweets. Performance is compared by measuring the accuracy of the predictions. We present and discuss a set of results achieved using different algorithms, which reveals some of the challenges of working with informal, short text such as tweets.

1. Introduction

Every text written by a person tends to show some characteristics that are unique to them as an individual, no matter how formal the text. This is the basis of the task of recognising author traits, which Zheng et al. (2006) showed to be so effective it could be used to identify authors of anonymous messages. However, are there characteristics that are common for specific *groups*? Are there words or phrases that are stereotypical of a younger or older person? How about gender? Answering these very questions was given as a task at PAN@CLEF 2016 (Rangel et al., 2016). Guided by the contestants' research and results, we explore different machine learning algorithms and present our findings on the most effective strategies for the given task.

We focus our training on a set of tweets given at PAN@CLEF 2016, which for each user gives their gender, age group, and a set of tweets written by them. For privacy reasons, their identity is hidden behind IDs that do not reveal any personal information. Due to the sensitive nature of collecting such data, we struggled to find additional datasets for testing, so we present results of training and testing over the corpus using 10-fold cross-validation.

2. Related Work

This is a relatively well-researched task, but since this was given as a topic at PAN@CLEF 2016, we focused on the approaches proposed by the contestants. A commonly used approach is using stylometry-based features, an example of which we find in (Ashraf et al., 2016). Encouraged by the results of previous research on stylometric approaches, such as in (Abbasi and Chen, 2008) and (Argamon et al., 2003), they create a similar model using a total of 56 stylistic features divided in three categories: *lexical features* such as average sentence or word length, *syntactic features* such as number of adjectives or verbs, various *vocabulary richness* measures, and *character based features* such as percentage of punctuation characters or character count. They have obtained promising results with an accuracy of 0.787 for gender and 0.983 for age on the training corpus, the same one available to us and used in our own research.

op Vollenbroek et al. (2016), the winners of the competition, gave somewhat lower accuracy in the development phase than the other mentioned competitors (0.457 for age and 0.707 for gender) but gave the most robust results across different text genres (trained on tweets, tested on blogs). They also used a stylometry-based approach with features designed to observe the small details that are characteristic of informal text, such as misspelled words, the use of a 'nose' in emoticons (i.e., ":)" vs. ":-)"), vocabulary richness, and others. They also included features that are characteristic of text in general, in order to prevent overfitting to a specific type of text since performance was graded across different genres of text.

We explore an alternative approach which does not use stylistic features, but rather relies only on vocabulary-based features: tf-idf weighted matrix representations of texts (Aizawa, 2003). This approach has been attempted by Agrawal and Gonçalves (2016), who achieved accuracy of 0.961 for gender and 0.642 for age. They outperform Ashraf et al. (2016) in gender, but give significantly weaker results for age group classification. We adopt their preprocessing and feature extraction method and explore different training algorithms in an attempt to push forward the accuracy for age, and possibly outperform the stylometry-based approach by Ashraf et al. (2016).

3. Data Processing

3.1. Preprocessing

Our corpus consists of xml documents containing tweets from various users. For each user, we are provided with labels of their gender ('FEMALE' or 'MALE') and their age group ('18-24', '25-34', '35-49', '50-64', '65-xx').

Following Agrawal and Gonçalves (2016), we preprocessed the tweets in several steps. First, we joined all the tweets of a single user into one document and extracted the raw text of the tweets (i.e., with no XML/HTML tags) using `lxml`.[1] Furthermore, we replaced all references to other users with @USERNAME, links to @LINK, emoti-

[1] `http://lxml.de/`

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 6–10, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

cons with @EMOJI, and numbers with @NUMBER using regular expressions. Numbers often had some trailing letters after it (i.e., "5th" became "@NUMBERth") which we removed. We also removed stop words, punctuation, duplicate tweets, and extra whitespace and converted all letters to lower case. Users that had no tweets associated with them were removed from the set of users.

We also created two alternative sets of preprocessed tweets with a slight variation - one with the stop words left in the text, and the other with both stop words left in and more specific tags for emoticons such as "@SMILE", "@FROWN", etc. The reasoning behind leaving stop words in instead of removing them as is standard practice with tf-idf matrices is that they might contain some important features for discerning the gender of the authors. Argamon et al. (2003) found that male-authored texts tend to have more determiners (*a, the, that, these*) and quantifiers (*one, two, more, some*), while female-authored text contained more pronouns (*I, you, she, her, their, myself, yourself, herself*), most of which are usually removed as stop words. However, their research was done using longer, more formal texts from the British National Corpus. This means that these differences might not show in our corpus, but nevertheless we included them for comparison.

3.2. Feature Extraction

After preprocessing, we extract the vocabulary and represent the documents as a tf-idf matrix using the `TfidfVectorizer` provided by `scikit-learn`[2] (Pedregosa et al., 2011). During this step, we ignore all words that appear only once in the corpus. This allows us to effectively ignore tweets from the corpus that were not in English, despite being tagged as such. After this step, we have a matrix of shape (420, 27128) which corresponds to 420 users and 27128 features, each representing a word in the vocabulary. For comparison, Agrawal and Gonçalves (2016) had 38267 features in the original matrix (before feature selection which will be discussed in Section 3.3.). This is a lot of features, many of which do not significantly contribute to the classification, so we reduce the dimension of this matrix as described in the next subsection.

3.3. Feature Selection

Here is where our work starts to diverge with (Agrawal and Gonçalves, 2016). They selected features based on information gain, i.e., they kept all features which had a non-zero information gain, separately for each classification. Instead, we use a `GradientBoostingClassifier` which enables retrieving features determined to be the most important while fitting. For this task, the parameters that gave the best results were: 1000 estimators, a maximum tree depth of 1, and a learning rate of 0.1. The other parameters were left as default. After feature selection, we are left with about 269 features for gender, and 582 features for age group classification. These numbers may vary due to the random nature of the classifier, so we fixed the random seed parameter in order to get comparable results. The following experiments are conducted over these reduced tf-idf matrices.

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

4. Experiments

We experimented with a number of different machine learning algorithms in order to explore how each one fares on identifying the gender and age of the tweets' authors. The models and its parameters we used are listed in Table 1. The parameters listed in the table are the ones we found to be optimal for that model using a grid search algorithm paired with 10-fold cross-validation. We also trained the model with all the original 27128 features with various algorithms. Since the reduced version consistently gave higher accuracy we included only the best result (in this case using linear SVC) of the tests using the original full set of features for comparison.

Additionally, in order to test our model over a different corpus, we searched for datasets that were labeled with the demographic data we are classifying. However, due to privacy concerns, such data is rarely made publicly and freely available, hence difficult to obtain. The additional dataset we found[3] and used for testing is similar to the one we used for training but much larger and from different users. However, there is only one tweet per user, which gives much shorter documents per user compared to our original dataset. It is also only labeled with gender, not age. Besides training with the original corpus and testing with the additional one ("AT1"), we also tried the opposite (i.e., training with the additional one, testing with the original one - "AT2"), and evaluating the model solely on the additional dataset using 10-fold cross-validation ("AT3"). For these three tests we used the model that gave the best results *for gender* in previous testing: ET1000 + SW. A joint vocabulary was built using the words that appear more than once in both datasets, and the number of features was reduced to 181 using the same method as in previous testing.

We evaluated our model by comparing the mean accuracy scores rounded to three decimal places with the ones reported in (Ashraf et al., 2016) - "ASH" and (Agrawal and Gonçalves, 2016) - "AGR" for training over the same corpus as ours. All results are listed in Table 2, sorted by their average score of gender and age classification accuracy. Each test is identified with the ID listed in the last column of Table 1 or in the text. The results using the first modified matrix (with stop words left in) are labeled as [ID]+SW. More detailed labeling of emoticons did not significantly improve results so they are not included in the table. Figure 1 shows a visualization of these same results. The results of experiments that include the additional dataset are shown in Table 3.

5. Analysis and Discussion

As we can see in Table 2 and Figure 1, we did not succeed in outperforming the *average* accuracy of (Ashraf et al., 2016) and (Agrawal and Gonçalves, 2016). However, almost all tests give a better result for gender classification than the first, and our top three classifiers outperform the latter in age. It seems that vector space models perform well on gender identification tasks, while stylometry-based models perform better on age classification. In (Lin, 2007)

[3] `https://www.kaggle.com/crowdflower/twitter-user-gender-classification`

Table 1: The list of algorithms and their parameters we tested.

Algorithm	Parameters	ID
Voting +		
log. regr., naive Bayes, multinomial NB	soft voting	VT1
log. regr., random forest, gaussian NB	hard voting	VT2
AdaBoost	n_estimators=50, DecisionTreeClassifier, lr=1.0	AB
Extra-trees classifier	1000 classifiers, criterion:entropy	ET1000
KNeighbors	5 neighbors, uniform weights, automatic algorithm	KN
LinearSVC	C=1, squared hinge loss, l2 penalty	LSVC
SVC	kernel:poly, C:10, gamma:100, degree:1	SVC

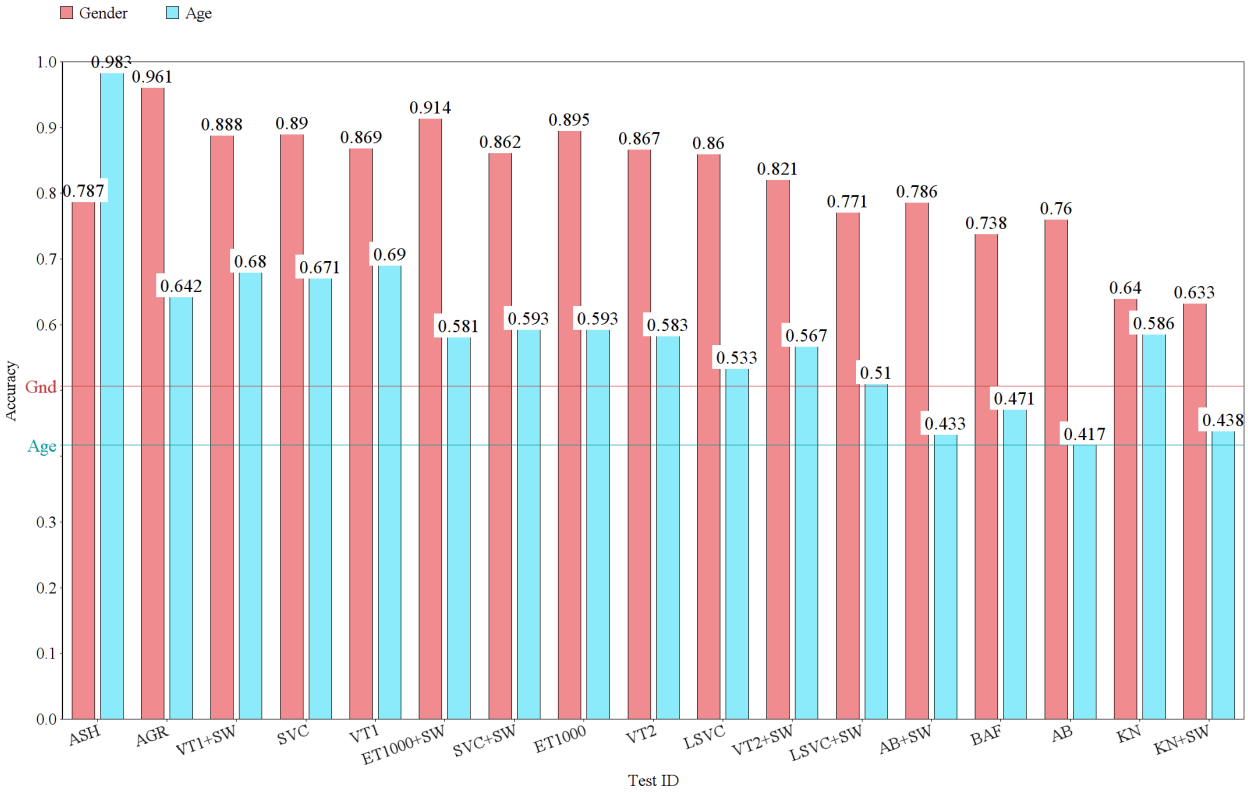


Figure 1: Graph of accuracy results of tests as listed in the table. The pink and blue horizontal lines represent the baseline accuracy of gender and age group identification, respectively. The baseline classifier always assigns the most common labels in our dataset: 'MALE' and '35-49'.

we find a possible explanation for this: gender differences tend to show in vocabulary. For example, a speech analysis study showed that male speech tends to be lexically richer and uses longer phrases, while female speech uses more verbs and shorter sentence structures (Singh, 2001). Interestingly, Lin (2007) finds that older women (50+) use a significantly richer vocabulary than the male counterpart. On the other hand, age difference is more prominent in stylistic features. Lin (2007) found that younger age groups user shorter sentences, while ages 30-39 use the longest sentences. Younger groups also tend to use less punctuation than older groups. This implies that older groups use a more formal writing style, which may include correctly using punctuation (e.g., no exaggerations such as "!!!!!"), capitalizing, and avoiding word abbreviations. It could also include using older versions of emoticons (e.g., ":-)" vs ":)") as observed by op Vollenbroek et al. (2016). All of these characteristics are represented if stylistic features are

Table 2: List of accuracy results of training and testing over the original corpus with the selected models. The best results and our own best results are bolded.

Test ID	Mean Accuracy		
	Gender	Age	Average
ASH	0.787	**0.983**	**0.885**
AGR	**0.961**	0.642	0.802
VT1+SW	0.888	0.680	**0.793**
SVC	0.890	0.671	0.781
VT1	0.869	**0.690**	0.780
ET1000+SW	**0.914**	0.581	0.748
SVC+SW	0.862	0.626	0.744
ET1000	0.895	0.593	0.744
VT2	0.867	0.583	0.725
LSVC	0.860	0.533	0.696
VT2+SW	0.821	0.567	0.694
LSVC+SW	0.771	0.510	0.640
AB+SW	0.786	0.433	0.610
BAF	0.738	0.471	0.605
AB	0.760	0.417	0.588
KN	0.640	0.586	0.563
KN+SW	0.633	0.438	0.536

Table 3: List of accuracy results of classifying gender using the new dataset.

Test ID	Mean Accuracy
AT1 - *train with orig., test with new*	0.531
AT2 - *train with new, test with orig.*	0.569
AT3 - *10-fold cv over new + best clf*	0.558

used, giving better results in the age group classification task.

The best performing model was the VT1+SW (soft voting + log. regr., naive Bayes, multinomial NB, including stop words). This model was the most similar to the one used by Agrawal and Gonçalves (2016), who used a stacking algorithm with the same base models, with one additional base model (Linear SVM). Similarly to our model, they had a large discrepancy between the average results of the gender and age classification tasks. This, along with the results of our other models that follow the same trend, shows that tf-idf matrices give good results in gender classifying tasks overall, regardless of the classifier used. On the other hand, it seems that such models are not as well suited for age identification, and instead should be done with a stylometric-based approach.

Table 4: Confusion matrices for tests AT1 (upper) and AT2 (lower), rows represent true labels, columns represent predicted labels.

	F	M
F	5465	1235
M	4809	1385
F	43	164
M	17	196

For gender identification in particular, the best performing model was the ET1000 + SW (Extra-trees classifier with 1000 classifiers, using stop words). This algorithm, like Random Trees algorithms in general, has shown to give competitive results for this kind of task which uses large datasets with a large number of features (Breiman and Cutler, 2007), so this result is not surprising.

Finally, we observed the confusion matrices for the tests where the classifier was trained and tested on different datasets, shown in Table 4. As we can also see in Table 3, in this case the results are barely over the baseline. It seems that when trained on the original corpus and tested on the newer one, the classifier tended to heavily choose to label tweets as female, whereas in the second test the opposite happens. We attempted to find a reasoning behind why this happens, however, as shown in Table 3, the new corpus by itself gives low accuracy when trained and tested using 10-fold cross-validation. We attribute this result to the fact that while in the original corpus there are about 1000 tweets per user, in the new corpus there is only one tweet per user. This shows that just one tweet does not give enough information to identify the author's gender or age, and if one wishes to attempt this task one should use as many tweets per user as possible in order to get accurate results.

6. Future Work

Based on our results, we predict that in order to maximize accuracy for both age and gender, one of two methods will give optimal results. The first would be using an ensemble model with two base models under it, one which is specialised for gender classification using tf-idf vectors, and the other for age classification using stylistic features. The meta-classifier would then select the more confident result as the final label. Alternatively, using both tf-idf vectors *and* stylistic features with one classifier may also prove to be effective. Finding the optimal balance between the weight of vector-space based features and stylistic features poses a challenge that we would like to pursue in our future research.

7. Conclusion

In this paper we explored various classification models over tf-idf vectorized tweets, in order to find the models which most accurately predict the age and gender of the authors. We found that the tf-idf approach shows generally good results for gender identification, with the best performing classifier being the Extra Trees Classifier. Age group identification was less successful, where stylometric approaches seem to perform better. This is because gender differences are more dependant on vocabulary, while age differences are found in stylistic features such as punctuation use, emoticon use, sentence length, and capitalization. We conclude that the best approach for joint classification of age

and gender would be a model that uses both tf-idf weighted vectors of text and stylistic features.

References

Ahmed Abbasi and Hsinchun Chen. 2008. Writeprints: A stylometric approach to identity-level identification and similarity detection in cyberspace. *ACM Transactions on Information Systems (TOIS)*, 26(2):7.

Madhulika Agrawal and Teresa Gonçalves. 2016. Age and gender identification using stacking for classification. *Notebook for PAN at CLEF 2016*.

Akiko Aizawa. 2003. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65.

Shlomo Argamon, Moshe Koppel, Jonathan Fine, and Anat Rachel Shimoni. 2003. Gender, genre, and writing style in formal written texts. *TEXT-THE HAGUE THEN AMSTERDAM THEN BERLIN-*, 23(3):321–346.

Shaina Ashraf, Hafiz Rizwan Iqbal, and Rao Muhammad Adeel Nawab. 2016. Cross-genre author profile prediction using stylometry-based approach. *Notebook for PAN at CLEF 2016*.

Leo Breiman and Adele Cutler. 2007. Random forests-classification description. *Department of Statistics, Berkeley*, 2.

Jane Lin. 2007. *Automatic author profiling of online chat logs*. Ph.D. thesis, Monterey, California. Naval Postgraduate School.

Mart Busger op Vollenbroek, Talvany Carlotto, Tim Kreutz, Maria Medvedeva, Chris Pool, Johannes Bjerva, Hessel Haagsma, and Malvina Nissim. 2016. Gronup: Groningen user profiling. *Notebook for PAN at CLEF 2016*.

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.

Francisco Rangel, Paolo Rosso, Ben Verhoeven, Walter Daelemans, Martin Potthast, and Benno Stein. 2016. Overview of the 4th author profiling task at pan 2016: cross-genre evaluations. *Working Notes Papers of the CLEF*.

Sameer Singh. 2001. A pilot study on gender differences in conversational speech on lexical richness measures. *Literary and Linguistic Computing*, 16(3):251–264.

Rong Zheng, Jiexun Li, Hsinchun Chen, and Zan Huang. 2006. A framework for authorship identification of online messages: Writing-style features and classification techniques. *Journal of the American society for information science and technology*, 57(3):378–393.

Simple Two-step Factoid Question Answering Based on Skip-gram Embeddings

Daniel Bratulić, Marin Kačan, Josip Šarić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{daniel.bratulic,marin.kacan,josip.saric}@fer.hr

Abstract

This paper presents a two stage factoid question answering system. Factoid question answering is the most researched task in question answering, where focus is on questions that require concise, factual answers. The dataset we used in our work comes from the TREC-9 QA task. In the first step - answer retrieval - for a given question we rank sentences based on their relevance to the question. In the second step, we extract the answer from the list of ranked sentences. The SVM model that we used for answer retrieval outperformed our own baseline that used *word2vec* vector similarity.

1. Introduction

Question answering is a branch of natural language processing focused on finding answers to questions from various knowledge sources. Factoid question answering focuses on short, specific questions that require the system to provide concise facts.

Our task was described in (Voorhes (2000)), though it differs slightly from it since the data [1] used in that task is not freely accessible. For this reason, the data we used was a fraction of all the sentences in those articles, that were roughly extracted with an automatic extraction system.

As the data obtained this way had many flaws, it was not ready to be given as input to our system. We had to do data preprocessing before the actual standard NLP preprocessing pipeline. Once we had prepared the data to a satisfactory degree, we were able to proceed with NLP preprocessing. All of these steps are described in Section 3.

In Section 4 we introduce a high-level system architecture that we had set out to implement during the task. The question answering process is split into two main steps. The first step is answer retrieval where we had to retrieve the most relevant sentences for each question and rank them by relevance. Other question answering systems usually retrieve documents or paragraphs, not sentences, but we only had sentences to work with. After retrieving the most relevant sentences, our system extracts one phrase from those sentences that best answers the given question.

For both steps we devised baseline methods so that we have a point of reference to later compare our advanced models and measure how much they improve upon the baselines. We present the evaluation of our system in Section 5.

Section 6 concludes the paper with a summary of our work, the insights we have gathered through working on this project and ideas for future improvements.

2. Related work

Factoid question answering is the most studied task in QA and many approaches have been proposed. In (Wang (2006)) a survey of answer extraction techniques is given. All QA systems use supervised methods to train models. Most of the approaches use question type information on the question side and information about named entities on the sentence side. Ittycheriah et al. (2001) proposed a statistical approach for this task, where they scored matching words using IDF. Researchers often rely on hand-crafted pattern matching in their answer extraction work. More recently, deep neural network (Iyyer et al. (2014)) models have become a popular approach to do question answering.

[1] http://trec.nist.gov/data/qa/t9_qadata.html

3. Preprocessing

3.1. Cleaning the data

As mentioned in introduction, the dataset we had at the beginning was not convenient to use to develop our models, until it had been refined by filtering all the nonsensical and meaningless sentences, words and special characters.

The data came in four parts:

1. **Questions** The set of all questions for which the answers were contained in other parts of the dataset. It consisted of about 600 questions.
2. **Trec9_sents** All the sentences intended to be used as the source of knowledge for our system to get the answers from. Initially, it contained about 210,000 sentences.
3. **Qa_judgments** Contained pairs of questions and excerpts of text labeled according to the relevance of the excerpt to the question. Labels *1* and *2* meant the excerpt was relevant to the questions, with the latter of the two labels denoting that the exact answer was not present in the excerpt. The label *-1* meant the excerpt was not relevant for the question. It has to be noted that, just by looking at these labeled pairs, we could find examples that were inconsistent with these given definitions.
4. **Patterns** Intended to be used to evaluate answer extraction, for each question, a list of *RegEx* patterns was given to check whether the phrase extracted by the system matches one of the patterns.

The issues we had with the data were the following:

- Blocks of text would start and end in the middle of sentences. Even if segmented correctly, this would give vague incomplete sentences.

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 11–15, University of Zagreb,
Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

- Often there were sequences of punctuation symbols, HTML tags and other special characters scattered throughout whole sentences.
- Sentences would often overlap, meaning that same relevant sentence would appear multiple times, either as a substring or a complete duplicate.
- Many sentences were huge blocks of text containing hundreds of words and other characters and tags. On the other hand, some sentences contained an unreasonably small number of words.
- Sentences used different notations for the same token. For example, the sequence "&apos" would represent an apostrophe. Such tags had to be replaced with proper characters.
- Sentences did not have unique identifiers. Rather, they had article tags by which they were referenced in the judgements file. Multiple completely different sentences could have the same article identifier.

We tackled these problems by firstly pushing the sentences through a parser, to try to segment large meaningful blocks into individual sentences. After that, we sorted the sentences by length and removed the both the unreasonably long and the unreasonably short sentences from the dataset.

To prepare our dataset for model training and evaluation, we had to map *trec9_sents* to *qa_judgements*. Because multiple sentences had the same identifiers and various near duplicates sentences were present in both files, we had to use approximate similarity measures to do the mapping.

To get the similarity between two sentences we took all of their character 3-grams to obtain a set of 3-grams for each sentence. Then we used Jaccard similarity measure to get the similarity between the two sets.

Jaccard similarity between two sets, S1 and S2, is defined as follows:

$$jaccard(S1, S2) = \frac{|S1 \bigcap S2|}{|S1 \bigcup S2|} \quad (1)$$

These preprocessing steps resulted in 30 questions not having a single sentence labeled as relevant for them. We removed these questions from consideration because, evaluated on them, the system could never get a reciprocal rank different from 0. In addition to that, there were 70 questions left where there is only one relevant sentence, and in general, a large part of questions had very few sentences labeled as relevant for them. This is shown in Figure 1.

Ideally, we would want a more balanced dataset, with more positive examples where our model could better learn to score the relevance of sentences.

There was another available file in the dataset called *ranked_list*. It consisted of scores for each article and corresponding question. As we worked with sentences instead of articles, we tried to reformulate the data from that file to be compatible with our dataset so we could use regression models for answer retrieval. While looking through the file and comparing it with *qa_judgements*, we found many discrepancies between labels of the two files and examples where, in our opinion, the ranks did not reflect the relevances well. Because of this, we were not optimistic about using regression. This is further discussed in Section 4.2.

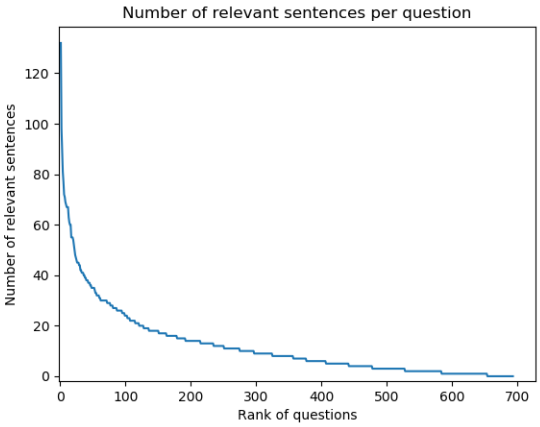


Figure 1: Figure which displays number of relevant sentences per question.

3.2. NLP preprocessing

After cleaning the dataset, we proceeded to with standard the NLP preprocessing pipline consisting of sentence segmentation, tokenization, lemmatization, stemming, part-of-speech tagging, named entity recognition and dependency parsing.

We used Spacy[2] for all steps except stemming, for which used the Porter Stemmer found in the NLTK[3] library.

4. System architecture

Our system consists of two main parts:

- relevant sentence retrieval and ranking
- answer extraction from retrieved sentences

Both parts can work and be evaluated individually. Also, the whole system can be evaluated by using the output of the first part as the input to the second part. A high-level scheme of the system is given in Figure 1.

Answer retrieval takes a question and all of the available sentences as input. It scores each of the sentences according to its relevance to the question and returns a sorted list of the highest scoring sentences.

Answer extraction takes the same question and the retrieved list to produce the phrase that represents the final answer to the question.

4.1. Question type classification

It is common for question answering systems to classify questions into coarse-grained type. Question type contains the information about the expected answer, so the system can use it to filter relevant sentences.

We chose to classify each question into one of five types: *agent*, *location*, *time*, *quantity*, *other*. All types are self-descriptive, with the *other* class being the universal class for all questions that do not fit into the first four types and that are more complex, asking for descriptions, reasons, definitions and more.

[2]https://spacy.io/

[3]http://www.nltk.org/

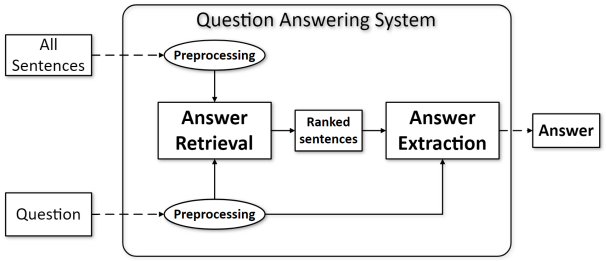


Figure 2: Architecture of the whole system for factoid question answering.

In our first attempt to classify questions, we assigned question types according to the question word appearing in the question. These are the rules that map the question word to question type:

- When - *time*
- Where - *location*
- Who - *agent*
- What, Which - *other*
- How much/many - *quantity*
- How long/old - *time*

These rules were used in both answer retrieval and answer extraction.

When we were developing the advanced answer extraction module, we noticed that around a half of questions from our dataset are questions with the question word *what*. Many of those questions could be classified into some of the other existing classes. For example, the question *"What is the real name of the singer, Madonna?"* is actually *agent*, and *"What city is Logan Airport in?"* is a *location* question.

We tackled this problem using the *word2vec* model. For every *what* question, we created its embedding vector representation as described in 4.2.1. Then, we would calculate that vector's similarity to vectors representing each of the five question types. We classified the question to the type corresponding to the most similar vector.

To get the *word2vec* representants question types, we took a sum of vectors representing the general words related to that type. For example, words we used to represent the *location* type are *where*, *place*, *city*, *state*, *region*, etc.

4.2. Answer Retrieval

4.2.1. Baseline model

As a baseline for relevant sentence retrieval and ranking we chose a relatively simple, yet powerful model. We used *word2vec* (Mikolov et al. (2013)) neural model to obtain word embeddings that capture the semantics of words. In this model, words are represented as real-valued 300-dimensional vectors. We did not create our own *word2vec* model. Instead, we used the pre-trained, publicly available Google's model[4]. To score a single sentence's relevance to a question, we would create vector representations of both the sentence and the question using the embedding vectors of the words contained in them.

[4]https://code.google.com/archive/p/word2vec/

We call this the *sent2vec* representation of a sentence, and it is calculated as a weighted sum of vectors of all the words in the sentence. Each vector is weighted by the inverse document frequency of the word throughout the *trec9_sents* dataset. This is a measure often used in information retrieval and it makes the more important words in the sentence have a greater impact in the sentence vector representation. In this context, we treat sentences as documents. The actual *sent2vec* formula is the following:

$$sent2vec(S) = \sum_{w \in S} word2vec(w) * idf(w) \qquad (2)$$

Once we have both the sentence and the question represented as vectors, we can measure their similarity as the cosine similarity of the two vectors. All available sentences are scored in this way and then sorted in a descending order. The final output of this part are the top 20 highest scoring sentences.

4.2.2. Advanced model

To improve upon the baseline, we decided to use supervised machine learning. Our dataset consisted of question-sentence pairs labeled with *1* and *-1* labels, denoting a relevant or an irrelevant sentence, respectively (we converted all *2* labels to *-1*). On the other hand, we needed real numbers as the output, not classes, in order to be able to rank all the sentences. Therefore, we chose to use a classification model and train it on labeled pairs. To obtain a relevance score for each sentence, we would take the signed distance from the model's decision boundary for that question-sentence pair. The higher the distance, the more confident the model is that the sentence is relevant.

We used a linear-kernel SVM implemented in the class sklearn.LinearSVC. It is an efficient implementation that can handle large training sets. The hyperparameters of the model are the regularization parameter *C* and the class weight. We optimized them via cross-validation.

We designed several features for our classifier model:

- Cosine similarity between our *sent2vec* sentences and questions. This feature was used as the baseline.
- Jaccard similarity between 3-gram sets of sentences and questions, as described in 3.1.
- Weighted sum of elements appearing in both the question and the sentence. The elements are weighted by the *idf* score. As elements, we consider tokens, lemmas and bigrams separately, to produce three different sums.
- Lengths of the sentence and the question encoded as one-hot vectors of size 4.
- Question and sentence type encoded as one-hot encoded vectors, as described in 4.3.1.

With all of the mentioned features, the dimensionality of our feature vector was 22.

As mentioned at the end of Section 3.1., we had a file *ranked_list* in our dataset that we tried to use to train an SVR (Support Vector Regression) model with the same features. As expected after observing the discrepancies between this file and the *qa_judgements* dataset, the regression model performed much worse than the classification model, so we discarded it early on.

4.3. Answer extraction

4.3.1. Baseline model

In our baseline answer extraction model, we firstly determined the question type as described in 4.1. Then, we found the first sentence that contained at least one named entity that corresponds to the question type. To do that, we needed to have a mapping from named entity types to question types. For example, we mapped the named entity types *person*, *organization* and *facility* to question type *agent*. No named entity types were mapped to the question type *other*.

After we found the sentence containing the correct named entity type, to produce the answer we simply extracted the first named entity of that type occurring in the sentence. If no such sentences were found, or if the question type was *other*, we returned the whole first sentence given as input.

4.3.2. Advanced model

For the advanced extraction model, we improved question type classification by using *word2vec*, as described in 4.1. Along with that, after filtering out the sentences whose type does not match that of the question, from all the remaining sentences we extracted all the named entities with type corresponding to question type. For each named entity we found, we calculated the weighted sum of its occurrences, with the weights being the score that answer retrieval assigned to the sentence where the named entity occurred. The named entity with the highest score is returned as the answer.

5. Evaluation

5.1. Answer retrieval evaluation

We trained and evaluated both models for answer retrieval on a closed collection of labeled question-answer pairs. Each question had on average 120 labeled sentences. We had considered including sentences outside the closed collection to train and evaluate the model with additional negative examples, but decided to discard the idea because the average number of negative examples for questions was already much larger than that of positive examples (see 1). Also, we did not know whether all the sentences outside the closed collection could be labeled as irrelevant.

As a performance measure we used the Mean Reciprocal Rank (MRR). For a single question, the reciprocal rank is calculated by taking the rank of the first correctly retrieved sentence and taking the reciprocal of that rank. Mean Reciprocal Rank is calculated by averaging the reciprocal rank for all questions.

We evaluated the models on 5 folds of the dataset so that we could perform a statistical test for the two developed models. We optimized the hyperparameters of the advanced retrieval model using a 3-fold cross-validation in a nested loop. The optimal hyperparameters found were $C = 2^{-13}$, $class\_weight = 300$.

We split the dataset into folds in a way that if two question-sentence pairs have the same question, they must either both be in the training set or in the testing set. We did the same when further splitting the training set for cross-validation. That way we ensure that the advanced model that uses machine learning is not evaluated on questions it has seen before.

The results of the evaluation are displayed in Table 1. From the results we can see that our advanced model significantly outperforms the baseline model at 0.01 level using Student's t-test.

Table 1: MRR scores for answer retrieval over 5 folds. † signifies statistical significance at level α=0.01.

Fold	Baseline	Advanced
1st fold	0.3772	0.4603
2nd fold	0.4547	0.4741
3rd fold	0.4315	0.4611
4th fold	0.3697	0.4667
5th fold	0.3679	0.4549
Mean over folds	0.4002	**0.4634**†

5.2. Answer extraction evaluation

We evaluated both models for answer extraction on the same splits of the questions in the dataset as in the case of answer retrieval. The retrieval model that supplied both extraction methods with relevant sentences was our advanced retrieval model, as it performed better than the baseline.

We measure the performance as the fraction of correct answers in total answers given.

The accuracy for the baseline model is: 8.7 %. The accuracy for the advanced model is: 9.6 %.

6. Conclusion and future work

This paper describes a system designed for the TREC-9 factoid question answering task. The system consists of two parts: answer retrieval and answer extraction. The answer retrieval step ranks sentences by their relevance to the given question using an SVM binary classifier's confidence score. The answer extraction step extracts the answer phrase that matches the question type. TREC-9 dataset was used for training and evaluation of models. In answer retrieval, our experiments showed that our advanced model outperforms the baseline model that uses only the *word2vec* similarity to rank sentences. Answer extraction proved to be a more complex task and our relatively simple rule-based model underperformed.

For future improvement, more focus should be put on researching and finding a more principled way to do answer extraction. One approach we considered is framing this task as a sequence labeling problem, where each word in a sentence would be labeled as being a part of the answer phrase or not.

Additionally, for retrieval, adding more complex syntax features, like the similarity between dependency parse trees of the sentence and the question could be beneficial.

References

Abraham Ittycheriah, Martin Franz, and Salim Roukos. 2001. IBM's statistical question answering system – trec-10.

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daum'e III. 2014. A neural network for factoid question answering over paragraphs.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

Ellen M. Voorhes. 2000. Overview of TREC-9 question answering track.

Mengqiu Wang. 2006. A survey of answer extraction techniques in factoid question answering.

Twitter Sentiment Analysis Using Word2vec And Three Level Sentiment Extraction

Valerio Franković, Mihovil Kucijan, Fran Varga

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{valerio.frankovic, mihovil.kucijan, fran.varga}@fer.hr

Abstract

This paper explores Sentiment Analysis in Twitter regarding tweet sentiment and sentiment towards a topic. To build an effective model three level sentiment extraction is used over preprocessed tweets. This feature extraction is combined with a skip-gram model based on $word2vec$ for topic-related tasks. The approach yields good results although it does not comply with the closed competition rules rather it only explores different methods one would use to build a model.

1. Introduction

The paper explores a solution to the task four given in SemEval 2016 competition. The task focuses on Sentiment Analysis in Twitter with three distinct subtasks. Subtask A is Tweet Polarity Classification, given a massage determine a positive, neutral or negative label. Subtask B and C respectively are Topic-Based Tweet Polarity Classification on a two point and a five point scale. In the world of today, social media has become the first platform people think of when they want to voice their opinion. Leveraging this enormous amount of data to predict the outcome of current events would be of great significance. Analyzing the global sentiment towards events encompasses a large area of Big Data science but in this paper we focus on analyzing the sentiment of a single tweet as a step in the process. A part of text posted on social media usually contain the posters feelings towards an event or topic, although news outlets and such also post on social media they are but a fraction of the corpus. Scaling the Tweet Polarity Classification to determine the general sentiment towards would be a strong tool. We preprocess tweets to build a good bag-of-words representation and try to capture as much of the features as we can. The manually extracted features include sentiment scoring and simple feature counting on three tweet levels: tweet, emoticon and hashtag level. We built a model using Multinomial Bayes classificator for the Subtask A, we used $word2vector$ representation with Multi-Layer Perceptron for the Subtask B and MLP Regressor which was then rounded for Subtask C. The tools were taken from the $sciki-learn$ and $nltk$ packages (Pedregosa et al., 2011; Loper and Bird, 2002)

2. Related Work

Sentiment analysis has been a hot topic for a while now. In the SemEval competition it is re-running for the 4th year in a row attracting more and more participants. The results and work can be seen in the Sem Eval task ranking papers (Nakov et al., 2016) and (Poursepanj et al., 2013).

Our approach is the most similar to the work of NRC-Canada (Mohammad et al., 2013). They use similar features such as negation scope, elongated words, sentiment score, capital words and others as well as n-gram and n-character bag-of-words representations passed to the SVM classifier to achieve great results. They have ranked first in the 2013 SemEval Competition (Poursepanj et al., 2013).

Word embedding is also a point of interest. The embedding is commonly used for Named Entity Recognition as in (Godin et al., 2015) but we show that it can be used for Sentiment Analysis. Vectors that model word context or distribution may be better suited for sentiment tasks than regular presence or $tf - idf$ vectorization.

Sarcasm detection could also be used to boost results. It would seem that sarcasm is the default way of communication in the young adult user base of Twitter so we hypothesize that work such as (Riloff et al., 2013) could be used to produce much better results.

Twitter as a database for sentiment analysis has been discussed at length and has been shown to be an interesting and non-trivial corpus (Pak and Paroubek, 2010).

3. Data And Metrics

Data we used is the SemEval 2013 and SemEval 2016 Sentiment Analysis GOLD dataset. For Subtask A the label distribution is given in Table 1. We conclude from the table that there exists a bias against negative tweets so accuracy is not a good score to use, this is why the F1 metric is used to evaluate.

We use the official 2016 task evaluation script as well as the $scikit - learn$ metrics. A similar distribution exists for B and C data sets. Recall and macro averaged Mean Absolute Error is used respectively for further tasks. The formula is as follows:

$$MAE^M(h, Te) = \frac{1}{|C|}\sum_{j=1}^{|C|}\frac{1}{|Tej|}\sum_{x_i \in Tej}|h((x_i) - y_i)|$$

The first sum is the macro averaging over the classes while the next sum is the usual sum of absolute errors for every sample in a class. Recall is a good metric for subtask B since only around 20% of the tweets are negative which means the Accuracy of Positive only Classifier would be around 80%. Recall is a better metric as it shows the percentage of True Positive out of all True, the same goes for Negative.

Macro averaged Mean Absolute Error over topics is used

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 16–19, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

Table 1: Data distribution.

Datasets	Strongly negative	Negative	Neutral	Positive	Strongly positive
Subtask A train	-	15.06 %	47.35%	37.59%	-
Subtask A test	-	15.76%	42.66%	41.58%	-
Subtask B train	-	17.37%	-	82.63%	-
Subtask B test	-	22.17%	-	77.83%	-
Subtask C train	1.48%	11.26%	27.96%	51.98%	7.32%
Subtask C test	0.67%	10.67%	48.86%	37.95%	1.85%

for Subtask C as averaging over classes gives us robustness against the heavy imbalance of the classes as less than 10% of the tweets belong to the highly negative and the highly positive class. Also, topic based averaging handles the topic distribution inequality. MAE is the score of error so in contrast to Recall and F1 lower MAE is better. Macro averaging is used as there exists a class imbalance. Calculating the scores over each class and then aggregating them gives the scores robustness and significance.

4. Methods

4.1. Subtask A: Tweet Sentiment Polarity Classification

4.1.1. Preprocessing And Feature Extraction

A tweet is processed in a way that lowers all the tweet letters, replaces emoticons and punctuation marks with certain words, stems those words and finally labels negative parts of the sentence which will be further explored in the next section, there is also an option to label words that are written with capital letters. We also strip hyperlinks and user references as we predict such entities do not bring sentiment, but rather are the target or the receiver of sentiment. We substitute these words for their generalized replacement, any @word is replaced with USER and any hyperlink is replaced by LINK. All other replacements are done similarly. To stem and lemmatize we used the $SnowballStemmer$ and $WordNetlemmatizer$ available as a part of the $nltk$ package (Loper and Bird, 2002). For the sentiment corpus we use the $SentiWordNet$ which is a part of the $nltk$ package.

The idea of our approach is that some features can be coded as words and the bag-of-words representation of the vector will include the wanted features without us manually extracting them. Manually extracted features would include simple counts of text we consider to be highly indicative of sentiment and sentiment corpus scores of the tweet on three levels.

4.1.2. Negation Scope

Finding the negation scope of NOT, or words ending in $N'T$ is the next part of preprocessing. The negative scope starts at the negation word and ends with a punctuation word eg. $STOP$. We add $'\_n'$ characters to the end of the words to denote their existence in the negative scope. This is a simplified approach of the NRC-Canada team (Mohammad et al., 2013).

4.1.3. Sentiment Extraction And Counting Features

The steps taken to extract the sentiment are lemmatization and POS-tagging as these are the keys to finding sentiment in the $SentiWordNet$ corpus. This is not included in the final processed tweet used to build the bag-of-words but only as a part of the sentiment extraction function. We extract sentiment on three levels: tweet, emoticon and hashtag level. We hypothesize that different parts of the tweet have a different correlation with the sentiment so we separate them. Tweet level sentiment includes all word in the tweet with the hashtagged word sentiment scaled with the factor of two to give it greater significance. We hypothesize that hashtags describe a tweet as a whole and therefore hashtagged words have more bearing on the sentiment of the tweet. Emoticon level and hashtag level scores do not include scaling. The features include the sum of sentiment over all words, the highest positive sentiment, the highest negative sentiment, and the absolute difference between negative and positive sentiment. We assume the last feature will model sarcasm as seen in (Riloff et al., 2013) to a certain degree. After Sentiment Extraction for every tweet we also count the number of elongated words number of hashtags and the number of capitalized words.

4.1.4. Text Vector Representation

The manually extracted features (sentiment scores and feature counts) are then concatenated to the end of the bag-of-words representation of the tweet. The BOW is constructed with the $scikit - learnCountVectorizer$ for the English language with the option for stop word removal and minimal corpus frequency turned on. $Tf - idf$ weighted features are not used as it has been shown they do not help a lot or at all (Pang and Lee, 2008). Only unigrams are used but some preliminary testing has shown that even bigrams could bring a lot more classification power. The best classifiers often use n-grams of many lengths as well as character sequences of varying length (Mohammad et al., 2013).

4.2. Subtask B And C: Topic Dependent Polarity Classification

For subtask B and C we took slightly different approach. Considering that subtask B and C classify sentiment towards certain topic, the sentiment expressed in tweet alone is not important if it is not directed toward the topic. Simplistic approach was chosen to tackle this problem using semantic similarity. To express semantic similarity we used dense representation of words using $word2vec$ neural model. SentiWordNet dictionary was used to capture

sentiment which has positive and negative score for each word not necessarily exclusive to each other. For our representation best model turned out to be multi-layer perceptron using regression for subtask C and classification for subtask B.

4.2.1. Preprocessing

For preprocessing we took similar steps as for task A, extracting smiles from tweets and replacing them with words but we did not stem the tweets since we are using word embedding model.

4.2.2. Word2vec

Core of our feature representation is $word2vec$ model. For these tasks we used pre-trained twitter model on 400 million tweets giving us robust word embedding model. The model has vocabulary of 3,039,345 words and its represented in 400 dimensional space. Training of the model was done using Skip-gram architecture and negative sampling for five iterations with context windows of one. If the frequency of the word was lower than five, the word was discarded (Godin et al., 2015). Using $word2vec$ model, tokenized tweet is then represented by averaging dense representation over each word in tweet. Same is done with given topic and concatenated to the tweet representation.

4.2.3. Additional Features

We have also added additional features by collecting information from sentiment lexicon SentiWordNet. SentiWordNet provides positive and negative sentiment for each word presented in dictionary. Using this we have added two features representing the sum of this positive and negative evaluation for each sentiment bearing word in tweet.

Similarly, we have added additional two features representing sentiment in an n-gram near the topic in the tweet. This was done to try to extract sentiment towards the topic specifically and recognize the difference between tweet level sentiment and topic level sentiment assuming that sentiment bearing words targeted towards topics are in close proximity to the topic. Given topic was searched in tweet by measuring semantic similarity using cosine similarity calculated as dot product of normalized vectors representing the words. Since topics are multi-word phrases for a given word in a tweet we calculated similarity by add cosine similarity to the power of two for each word in topic. This way we gave preference to the words that are similar to a higher degree with specific word in a topic then to the words that are closely similar to all. After that we added up sentiment for words found in an n-gram around the topic.

5. Evaluation

5.1. Subtask A: Tweet Sentiment Polarity Classification

To classify the data we used Multinomial Naive Bayes classifier. Nested cross validation was run over 10 folds in the outer loop with three folds in the inner loop. The results on the 2013 data set are a F1 score of 65% and for 2016 a F1 score of 48%. Ranking our model at the official rankings of the competition we fall at 18th place in the lower middle rankings for 2013 and at the 26th place in the 2016 rankings. The results are not the best but this was to be expected we only used simple features and BOW representations while the best ranking teams use n-grams in the range of one to six with char sequences and a feature vector extracted from a number of corpora. To measure the significance of our model we compared it to the base model which does not use any preprocessing, only BOW representation. We ran a Student's t-test and which has shown that our preprocessing shows significant improvement over the baseline at 0.001 level. The system used was constrained by the hardware we did not want a system too complicated and power demanding system. We expect that a SVM classifier with a more extensive Grid Search would be able to achieve even better results since it is more robust and better at high dimension classification (Mohammad et al., 2013). All the models were taken from $scikit-learn$ (Pedregosa et al., 2011).

Table 2: Subtask B results.

Models	recall-macro	accuracy score
baseline	0.389	0.778
final model	0.829	0.865

Table 3: Subtask C results.

Models	MAE macro	MAE	accuracy score
baseline	0.585	0.589	0.479
final model	0.525	0.509	0.510

5.2. Subtask B And C: Topic Dependent Polarity Classification

The evaluation for subtasks B and C was done using same metrics as proposed on SemEval2016, as previously noted in this work. For a given representation of tweets we have used Multilayer Perceptron using classification on subtask B and MLP Regression for subtask C. Rounding was used on the regressor to label a value to the closest class. The parameters were chosen by nested cross validation as described before. This resulted that the final model has only one hidden layer because any more layers would overfit the data. The results are shown in Table 2 and Table 3.

For the baseline we used the same model with only $word2vec$ representation of tweets. On subtask B this resulted in classifying the test data to the most prevalent class. Our final model outputs a significantly higher score. When comparing with the SemEval2016 rankings, it would be ranked first place. Considering it is a rather simple model most probable reason for such good results is a very robust and large $word2vec$ model trained on a large Twitter corpus which gives a very accurate representation and correctly identifies topics in tweets.

For subtask C our baseline scored quite well comparing to SemEval2016 results, showing the significance of a good $word2vec$ model, despite that adding additional features

and topic representation increased the score even further showing that more information can be gained using semantic similarity of $word2vec$ and predefined word sentiment.

6. Conclusion And Further Work

Word embedding is a very robust and simple to use tool for Sentiment Analysis. We have shown that a pre-built Twitter word embedding gives the majority of the classifiers decision, but we have also shown that our preprocessing is significant. The results on Subtask A are not as great but it is to be expected, we did not use word embedding for this task or the high dimension representation many other teams used, but still scored acceptable results. For further work more complex models using the features extracted could be explored, as well as structural representation of tweets to determine sarcasm and other hidden features.

References

Fréderic Godin, Baptist Vandersmissen, Wesley De Neve, and Rik Van de Walle. 2015. Multimedia lab@ acl w-nut ner shared task: named entity recognition for twitter microposts using distributed word representations. *ACL-IJCNLP*, 2015:146–153.

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*.

Saif M Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. *arXiv preprint arXiv:1308.6242*.

Preslav Nakov, Alan Ritter, Sara Rosenthal, Fabrizio Sebastiani, and Veselin Stoyanov. 2016. Semeval-2016 task 4: Sentiment analysis in twitter. *Proceedings of SemEval*, pages 1–18.

Alexander Pak and Patrick Paroubek. 2010. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10.

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135.

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.

Hamid Poursepanj, Josh Weissbock, and Diana Inkpen. 2013. uottawa: System description for semeval 2013 task 2 sentiment analysis in twitter. *Atlanta, Georgia, USA*, page 380.

Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. 2013. Sarcasm as contrast between a positive sentiment and negative situation. In *EMNLP*, volume 13, pages 704–714.

How Deep Is Your Humor? Ranking Humor Using Bi-Directional Long-Short Term Memory Networks

Bartol Freškura, Filip Gulan, Damir Kopljar

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{bartol.freskura,filip.gulan,damir.kopljar}@fer.hr

Abstract

In this paper, we consider the task of comparative humor ranking in two manners: detecting which tweet of two is more humorous and ranking the given tweets by how humorous they are in three classes. We opted for a different approach based on recent deep neural models in order to eschew manual feature engineering. Our model used bi-directional *Long Short Term Memory* networks and convolutional neural networks, in combination and separately, We based our feature vector construction on readily available pre-trained Twitter *GloVe* word embeddings along with learned character embedding. The system was trained, tuned, and evaluated on the SemEval-2017 Task 6 dataset for which it yields ambitious results.

1. Introduction

Understanding humor expressed in text is a challenging Natural Language Problem (NLP) which has not yet been addressed extensively in the current AI research. Humor is often subjective and relies on the vast knowledge base which is sometimes hard to reason even for humans. It is also important to note that what is humorous today might not be humorous tomorrow due to the fact that humor can be trend dependent.

In this paper, we describe a system for humor comparison and ranking. Our system is designed to solve two tasks. For the first task, further referred as Task A, the system is given two tweets and with a goal of predicting the funnier tweet. For the second task, further referred as Task B, the system is given a set of tweets and it should rank them in three categories (2 for the most humorous tweet, 1 if the tweet is in the top ten humorous tweets, and 0 otherwise).

To learn and test our model we used a novel dataset that was created as a part of the SemEval-2017 Task 6 (Potash et al., 2016). The dataset consists of the tweets viewers sent as a part of the Comedy Central show @midnight[1]. For every episode, a topic was defined and viewers were asked to send humorous tweets on the given topic.

2. Related work

In the last few years, there were many approaches in humor detection (Mihalcea and Strapparava, 2005; Reyes et al., 2013; Zhang and Liu, 2014; Barbieri and Saggion, 2014; Yang et al., 2015). Some of those works (Reyes et al., 2013; Zhang and Liu, 2014; Barbieri and Saggion, 2014) have also acquired humor dataset from Twitter. Most of the related works separate the apprehension of humor into two groups: humor and non-humor, basically a binary classification. This representation ignores the continuous nature of humor, while also not accounting for the subjectivity in perceiving humor (Potash et al., 2016).

Donahue et al. with *HumorHawk* team have won the contest on Task A (no data available for Task B). Their system utilizes recurrent deep learning methods with dense embeddings. They used *GloVe* embeddings in combination with a novel phonetic representation which is used as input to LSTM. After LSTM, they have stacked character-based CNN model and an XGBoost (Chen and Guestrin, 2016) component in an ensemble model which achieves 0.675 accuracy on the evaluation data for the Task A.

Kukovačec et al. with *TakeLab* team have participated in both A and B tasks and have won the second place in both of them. Unlike the other two teams (Donahue et al., 2017; Baziotis et al., 2017) who used deep learning models with automatic feature extraction, they used a manually handcrafted rich set of features: cultural reference features, binary and count-based features, readability-based features, humor-specific features and sentiment-based features. As a classifier, they used gradient boosting model which achieves 0.641 accuracy on the evaluation data for the Task A, and a distance of 0.908 on Task B evaluation data (lower is better with the worst score equal to 1).

Baziotis et al. with *DataStories* team have won the third place on Task A (no data available for Task B). They used a Siamese architecture (Bromley et al., 1993) with bi-directional LSTMs, augmented with an attention mechanism. Their model leverages word embeddings trained on a huge collection of unlabeled Twitter messages. The system achieves 0.632 accuracy on the Task A evaluation data.

3. Architecture

In this section, we describe the architecture of our system. Our most complex architecture consists of the bi-directional *Long Short Term Memory* (Hochreiter and Schmidhuber, 1997) layer, further referred as the Bi-LSTM, a convolutional layer (Lecun et al., 1998), further referred as the CNN, and a fully connected layer with a softmax layer at the end for predicting class probabilities.

The whole pipeline was built using the open source frameworks *TensorFlow* [2] (Abadi et al., 2015) and *scikit-learn* [3] (Pedregosa et al., 2011).

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

[2] https://www.tensorflow.org/

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

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 20–23, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

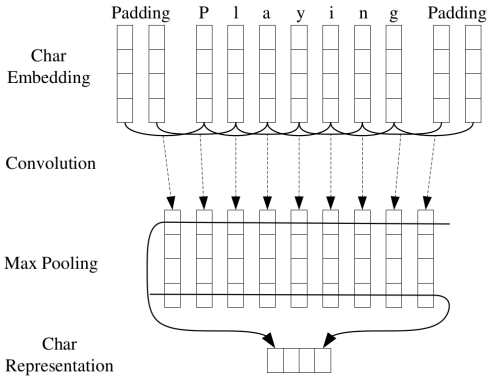


Figure 1: Word embeddings and the CNN architecture, from (Ma and Hovy, 2016).

3.1. Recurrent neural networks

The main idea behind RNNs lies in retaining information from "history". In the context of NLP, history refers to observing the context of the sentence up to the currently processed word. Despite the promising results in short sentences, RNN losses its performance dramatically with the increasing sentence length due to the gradient vanishing (Bengio et al., 1994) and exploding problems (Pascanu et al., 2013). LSTMs were designed with the purpose of correcting the RNNs shortcomings. Although LSTM can successfully capture the past context, it is sometimes good to have an insight into the future sentence context. Bi-LSTMs model implements this by adding an extra LSTM layer which has a reversed information flow meaning that the information is propagated from the end of the sentence towards the beginning. The output of a Bi-LSTM is a concatenated vector of the two opposite LSTM layers.

3.2. Convolutional neural networks

CNN networks are famous for their appliance in the *Computer Vision* domain but have also demonstrated an ability to extract morphological information from word characters, encoding them into neural representations. We first create a hash map of all characters that appear in the dataset where values are arbitrarily assigned integer values. All sentence characters are then represented using their mapped integer values but the padding is also applied on the word level as shown in Figure 3.2.. Encoded sentence represents an input which is fed into a trainable character embedding layer of $C_e \times V$ dimensions, where C_e is the character embedding size, and V is the number of unique characters in the corpus.

3.3. Used models

To test how every architecture handles humor detection we have created three models based on the following architectures: CNN, Bi-LSTM, and a combination of the former two.

CNN network Previous work (Potash et al., 2016) has demonstrated that CNNs are top performers when it comes to capturing humor. Our first model consists of the character embedding layer followed by a 1-D CNN layer for extracting features. CNN tweet features are joined and fed into a fully-connected layer with a softmax layer for predicting classes.

Bi-LSTM network Our second model is a pure bi-directional recurrent model with LSTM cells. As input features we use 100 dimensional *GloVe* [4] (Pennington et al., 2014) vectors trained on the $27B$ Twitter dataset. Fully connected and softmax layers are also placed at the networks end.

CNN + Bi-LSTM network To use the best of each of the convolutional and recurrent networks, we construct a merged network consisting of word embeddings, CNN, Bi-LSTM, and a fully connected layer. We hope this network can capture inter-word relations using the Bi-LSTM module as well as the inter-character features with the CNN layer.

4. Dataset

We trained and evaluated models on the dataset that was created from tweets of viewers who watched TV show @midnight. As part of this *game-show* viewers were asked to write a humorous message about a topic that was announced in the show. The day of the ensuing episode, @midnight would create a post that announced the top-10 tweets from the previous episode. The whole dataset consists of 11,685 tweets about 106 different topics. Every tweet in the dataset is labeled with one of the three labels. Label 2 indicates the winning tweet, label 1 a top-10 tweet, and label 0 is intended for all other tweets. A number of tweets per topic vary among different topics, but 71% of topics contain at least 90 tweets. Topics with the lowest number of tweets have around 20 tweets, and topics with the highest number of tweets have around 180 tweets. It is also important to note that for some topics like *FastFoodBooks* external knowledge is required to understand the joke, while for others like *IfIWerePresident* it is not.

Following prior work (Donahue et al., 2017; Kukovačec et al., 2017; Baziotis et al., 2017), we decided to tackle both of tasks with the single base model. For the task A, we have implemented the system which on its input accepts two tweets and predicts which tweet of the given two is more humorous. For the task B we used the same model in voting manner. More specifically, we created all possible pairs of all tweets for given hashtag and fed them to the described system. Then we counted how many times a tweet was more humorous than other tweets, and ranked the tweets by that number.

5. Experiments

We divide this section into two parts: the former one reporting results on the non-evaluation data, and the latter results on the official evaluation data.

5.1. Model optimization

During the model training and evaluation, we used a k-fold cross-validation technique ($k = 35$) to properly optimize our models hyper-parameters. Grid search method was not

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

Table 1: Final network hyper-parameters.

Hyperparameter	Value
Dropout rate	0.5
BLSTM hidden state	128
Sentence timestep	30
Learning rate	0.0002
CNN filter size	60
FC layer size	256
Character embedding layer size	50

Table 2: 95% confidence scores on the non-evaluation data for all models (in $\%$).

	Baseline	Bi-LSTM	CNN	CNN + Bi-LSTM
Accuracy	50.1 ± 0.2	$\mathbf{67.8 \pm 1.7}$	52.9 ± 1.4	67.0 ± 1.7
Precision	49.8 ± 0.3	$\mathbf{68.2 \pm 1.7}$	54.1 ± 1.4	67.4 ± 1.7
Recall	50.3 ± 0.2	$\mathbf{67.8 \pm 1.7}$	52.9 ± 1.4	67.0 ± 1.7
F1	50.0 ± 0.2	$\mathbf{67.8 \pm 1.7}$	53.2 ± 1.4	67.0 ± 1.7

feasible in our case due to large parameters search space and slow training time. Our method was to change each parameter value from the low to high extremes and see how it affects the model performance, in the end interpolating to find near optimal parameters. In addition to using dropout for regularization, we also employed the early-stopping (Caruana et al., 2000) method to achieve the best possible validation performance. Table 1 shows the final hyper-parameters used in training and validation. For the optimization algorithm, we used Adam (Kingma and Ba, 2014).

Our model is trained to maximize the class probability $p(y|x)$ using cross-entropy as the loss function. Output 1 means that the first tweet was funnier and 0 otherwise. In our results, we report results in form of four metrics: accuracy, precision, recall and F1 score. All entries represent 95% confidence intervals calculated from the 35 k-fold validation runs. Each of the runs was trained for only one epoch due to severe overfitting problems after the first epoch.

In Table 2 are the results from the non-evaluation data. Baseline model randomly guesses the funnier tweet and is expected to have metrics around 50%. We can see that the Bi-LSTM model performs the best so we further reference it as our best model.

5.2. Evaluation data results

In this section, we demonstrate how our best model, called *EuroNeuro*, compares with other solutions on the official evaluation data. Note that the accuracy and distance measurements listed in Table 3 and Table 4 are defined by the task organizers (Potash et al., 2016).

Our top model outperforms the best result in Task A by a margin of 1.6%, placing our model at the top of the list. In Task B, our model ranks second which is still competitive because the best team in Task A does not even have top results in Task B, meaning our model can perform well on both tasks.

Table 3: Official Task A results in comparison with our Bi-LSTM model.

Team	Accuracy
EuroNeuro	**69.1**
HumorHawk	67.5
TakeLab	64.1
HumorHawk	63.7
DataStories	63.2
Duluth	62.7

Table 4: Official Task B results in comparison with our Bi-LSTM model (lower is better).

Team	Distance
Duluth	0.872
EuroNeuro	**0.881**
TakeLab	0.908
QUB	0.924
QUB	0.924
SVNIT@SemEval	0.938

6. Conclusion

We proposed three different models for solving comparative humor ranking tasks of pairwise comparison and direct ranking classification. All three models use deep learning architecture by combining approaches of recurrent and convolutional neural networks.

For pairwise comparison task best results were achieved using the Bi-LSTM model result in 69.1% accuracy score on unseen evaluation data, and for direct ranking classification task best results were achieved using same Bi-LSTM model and were 0.881 on unseen evaluation data. Model evaluation on final unseen data is done using official evaluation scripts given in SemEval-2017 Task 6.

We have compared our results with the results of other task participants resulting in our model taking the first place on the Task A, and ranking second on the Task B. The main distinction between our model and competitive models is the lack of hand engineered features which indicates that automatic feature extraction using deep learning framework has a great prospect in this task and requires further work.

For the next step, we would experiment with specially adapted word embeddings trained only on the humor containing corpus. We believe it is crucial for word vectors to learn semantic meaning from the domain specific data because of the complex humor structure.

Acknowledgements

We thank TakeLab[5] team for providing us with the adequate computing resources.

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

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Francesco Barbieri and Horacio Saggion. 2014. Automatic detection of irony and humour in twitter. In *International Conference on Computational Creativity*, Ljubljana, Slovenia.

Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. Datastories at semeval-2017 task 6: Siamese lstm with attention for humorous text comparison. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 381–386, Vancouver, Canada, August. Association for Computational Linguistics.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 737–744, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Rich Caruana, Steve Lawrence, and Lee Giles. 2000. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *NIPS*, pages 402–408.

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

David Donahue, Alexey Romanov, and Anna Rumshisky. 2017. Humorhawk at semeval-2017 task 6: Mixing meaning and sound for humor recognition. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 89–93, Vancouver, Canada, August. Association for Computational Linguistics.

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

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

Marin Kukovačec, Juraj Malenica, Ivan Mršić, Antonio Šajatović, Domagoj Alagić, and Jan Šnajder. 2017. Takelab at semeval-2017 task 6: #rankinghumorin4pages. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 387–391, Vancouver, Canada, August. Association for Computational Linguistics.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Rada Mihalcea and Carlo Strapparava. 2005. Making computers laugh: Investigations in automatic humor recognition. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 531–538, Stroudsburg, PA, USA. Association for Computational Linguistics.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318.

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

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

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

Antonio Reyes, Paolo Rosso, and Tony Veale. 2013. A multidimensional approach for detecting irony in twitter. *Language Resources and Evaluation*, 47:239–268.

Diyi Yang, Alon Lavie, Chris Dyer, and Eduard H. Hovy. 2015. Humor recognition and humor anchor extraction. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *EMNLP*, pages 2367–2376. The Association for Computational Linguistics.

Renxian Zhang and Naishi Liu. 2014. Recognizing humor on twitter. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 889–898, New York, NY, USA. ACM.

Semantic Textual Similarity Using SVM and Deep Learning

Bruno Gavranović, Neven Miculinić, Stipan Mikulić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{bruno.gavranovic, neven.miculinic, stipan.mikulic}@fer.hr

Abstract

In this paper we present our work on Semantic Textual Similarity (STS) problem during Text analysis and retrival (TAR) class at FER. STS measures semantic similarity between two sentences. We use two approaches: an SVM model with feature preprocessing and a deep learning model trained on a language modelling task. Datasets were obtained using human annotators and their average semantic similarity score is given. We use Pearson correlation coefficient in our analysis and achieve 0.7795 on our test set respectively.

1. Introduction

```
Two dogs play in the grass.
Two dogs playing in the snow.
```

At the heart of STS, semantic textual similarity, is assigning numerical score on two text similarity. By text, it could mean full document, paragraph, or in this paper's case only one sentence. Beginning quote showcases one such sentance pairs, where human annotators rated 2.8 semantically similar on 0-5 scale, where 0 mean full semantic dissimilarity, and 5 full equivalence with various shades in between. STS developed system and techniqus have many further applications, transfer learning of sorts. Machine translation(MT), Summarization, Generation and Question Answering(QA) are some of them. Often new techniques invented in STS context generlize to earlier mentioned domains, as well as NLP fiels as a whole.

2. Related Work

STS has short and fruitful history, one with many ideas flying around. In its current form it appeared in 2012, on SemVal (Agirre et al., 2012) as task 6. In 2012, the best system (Bär et al., 2012) used lexical similarity and Explicit Semantic Analysis(ESA) (Gabrilovich and Markovitch, 2007). Following year Latent Semantic Analysis(LSA) (Deerwester et al., 1990) model (Han et al., 2013) with additional external information sources, WordNet and n-gram matching technique.

Following two years (Sultan et al., 2014) and (Sultan et al., 2015) dominate the competition with new algorithm – they align the words between new sentences. Other notable approaches come from logic side, its representative paper being (Beltagy et al., 2014).

3. Extent of the Paper

Our contribution consists of implementation of two machine learning models: SVM model and a deep learning model.

3.1. Data Analysis

We used dataset from SemEval 2017 Task 1[1]. The dataset has 250 instances. Each instance consists of two sentences and our tash is to predict semantic similarity between them. In order to get some deeper insights on data we did some minor data analysis. We got to conclusion that we are dealing with very short text. Also, we concluded that the smaller difference between lengths of sentences is, the more similar they are. This can be seen clearer on following plots.

Table 1: Stats about lengths of sentences.

	Without stopwords	With stopwords
Sentence length	33.004	43.838
Tokens in sentence	5.488	8.702

3.2. Baseline Models

To get an idea of how well our models perform, we used two baseline models. Neither of them is a machine learning model; in other words, they do not learn from data, but are rather simple statistics measures.

3.2.1. Jaccard Similarity Baseline

One of the simplest approaches to measuring semantic similarity is the Jaccard index. It is a statistic used for comparing similarity and diversity of finite sets. Let $J(S_1, S_2)$ represent the Jaccard index between two sentences S_1 and S_2. The similariy between sentences is defined as:

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}, \tag{1}$$

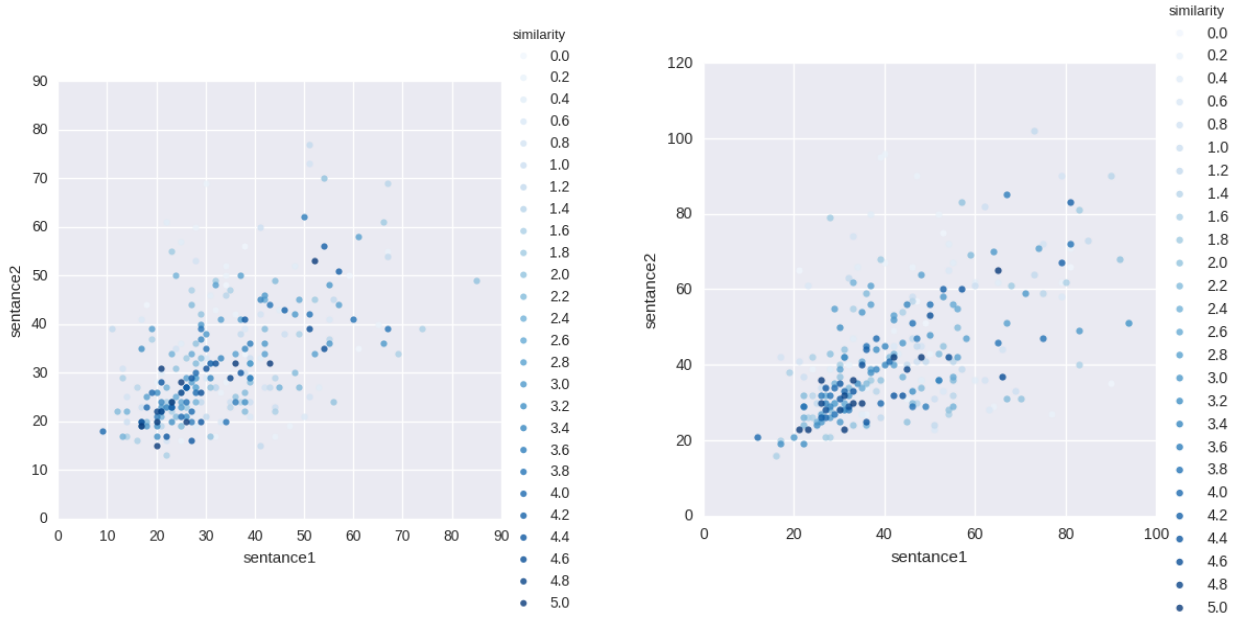
where the elements of S_1 and S_2 are their words, in lowercase.

3.2.2. Cosine Similarity

Unlike in the encodings of images, necessary information to decode the data is *not* stored in the encodings of words. Natural language is encoded in arbitrary sequences of symbols and provides no useful information to the model regarding the relationships that may exist between various concepts. One method that is effectively captures the semantic meaning behind the various concepts are the word embeddings.

[1] http://alt.qcri.org/semeval2017/task1/

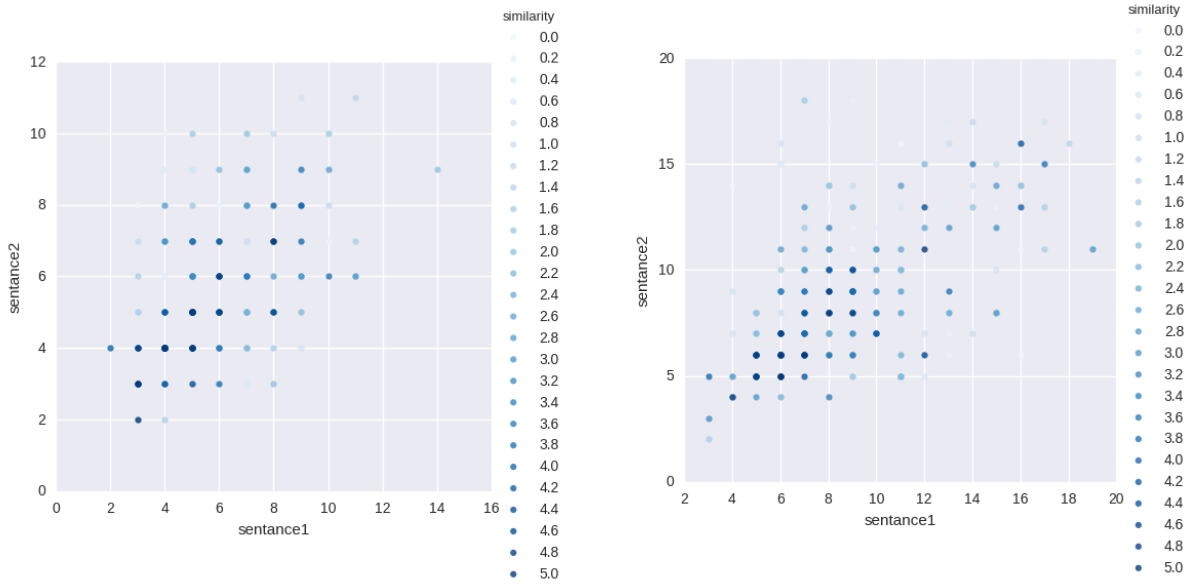
Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 24–29, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.



(a) Sentence lengths without stopwords

(b) Sentence lengths with stopwords

Figure 1: Sentence length stats



(a) Number of tokens without stopwords

(b) Number of tokens with stopwords

Figure 2: Number of tokens stats

Simple approach to extending the word embedding system to a sentence is simply performing some aggregation operation on each of the embeddings and then performing the cosine similarity measure. We used the summation function aggregation on the words of the sentence, after removing stopwords. Stopwords list is used from the NLTK library. Word embedding used is the English version of the FastText word embeddings (Joulin et al., 2016).

3.3. SVM

3.3.1. Features

All of the features we used are from (Saric et al., 2012) paper.

Ngram Overlap

Let S1 and S2 be the sets of consecutive ngrams in the first and the second sentence, respectively. The ngram overlap is computed for unigrams, bigrams, and trigrams. It is defined

as follows:

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

The ngram overlap is the harmonic mean of the degree to which the second sentence covers the first and vice versa.(Saric et al., 2012)

WordNet-Augmented Word Overlap
In order to determine some semantical meaning from words we define the WordNet augmented coverage $PWN(\cdot, \cdot)$:

$$P_{WN}(S_1, S_2) = \frac{1}{|S_2|} \sum_{w \in S_1} score(w, S_2) \quad (3)$$

$$score(w, S) = \begin{cases} 1 & \text{if } w \in S \\ \max\limits_{w' \in S} sim(w, w') & \text{otherwise} \end{cases} \quad (4)$$

where $sim(\cdot, \cdot)$ represents the WordNet path length similarity. The WordNet-augmented word overlap feature is defined as a harmonic mean of $PWN(S_1, S_2)$ and $PWN(S_2, S_1)$.(Saric et al., 2012)

Weighted Word Overlap
We define information contente measure to give some words more importance:

$$ic(w) = ln \frac{\sum_{w' \in C} freq(w')}{freq(w)} \quad (5)$$

where C is the set of words in the corpus and freq(w) is the frequency of the word w in the corpus. Since we got very small dataset we used *nltk brown* dataset to calculate word frequency distribution. The weighted word coverage of the second sentence by the first sentence is given by:

$$wwc(S_1, S_2) = \frac{\sum_{w \in S_1 \cap S_2} ic(w)}{\sum_{w' \in S_2} ic(w')} \quad (6)$$

where S_1 and S_2 are words in sentences.
The *weighted word overlap* between two sentences is calculated as the harmonic mean of the $wwc(S_1, S_2)$ and $wwc(S_2, S_1)$.(Saric et al., 2012)

Number of tokens difference
Difference between number of words per sentence is defined as:

$$diff(S_1, S_2) = abs(|S_1| - |S_2|) \quad (7)$$

where S_1 and S_2 are words in sentences.

Vector Space Sentence Similarity
We define each sentence as vector $u(\cdot)$ by summing all word embeddings in the sentence S: $u(S) = \sum_{w \in S} x_w$ where x_w is word embedding for each word. Another similar representation $u_w(\cdot)$ uses the information content $ic(w)$ to weigh the word embedding vector of each word before summation: $u_w(S) = \sum_{w \in S} ic(w) \cdot x_w$. We use $|cos(u(S_1), u(S_2))|$ and $|cos(u_w(S_1), u_w(S_2))|$ for the vector space sentence similarity features. (Saric et al., 2012)

Table 2: Averaged scores.

	R2	Pearson
score	0.5898	0.7795
std	0.011393	0.006625

Shallow NER Features
For this feature we simply count all words that are capitalized.

Numbers Overlap
For numbers overlap we define following three feature: $log(1 + |N_1| + |N_2|)$, $2 \cdot |N_1 \cap N_2| / (|N_1| + |N_2|)$ and $N_1 \subseteq N_2 \vee N_2 \subseteq N_1$ where N_1 and N_2 are sets of numbers in two sentences. We treat all numbers as decimal numbers.

3.3.2. Model
All features were preprocessed to zero mean and unit variance. For selection of best model we used nested cross validation, with Grid search in inner folds. Both inner and outer CV used 5-Fold. We optimized following parameters of SVR model:

$$kernel = \{linear, rbf\}$$
$$C = \{2^{-7}, 2^{-6}, ..., 2^{6}\}$$
$$gamma = \{2^{-5}, 2^{-4}, ..., 2^{2}\}$$

3.3.3. Evaluation
Since we used nested cross validation evaluation metrics are averaged over all testing folds. We used R2 and Pearson correlation as evaluation metrics. Nested CV scores are dispalyed on following plots:

3.4. Deep Learning

3.4.1. Motivation
Although feature engineering measures are able to provide an approximate measure of similarity, it is trivial to reverse-engineer adversarial examples that completely break the system. Consider, for example, two following sentences:

```
Compared to me, he's heavier.
I have less kilograms than him.
```

Although their Jaccard index is zero (they share no common words), their semantic similarity is high.

Similar examples can easily be generated by replacing any specific word with its definition:

```
Backpropagation.
Reverse-mode automatic differentiation.
```

Concepts on a such high level of abstraction easily break even the strongest systems since new word definitions could be defined arbitrarily. No known models are able to generalize well to strong adversarial examples in an arbitrary language. Such high level generalization could be considered as actual *understanding* of the natural language, which is an AI-complete problem.

A step forward to a model which could provide strong generalization capabilities is the one that has little to none

Figure 3: R2 score

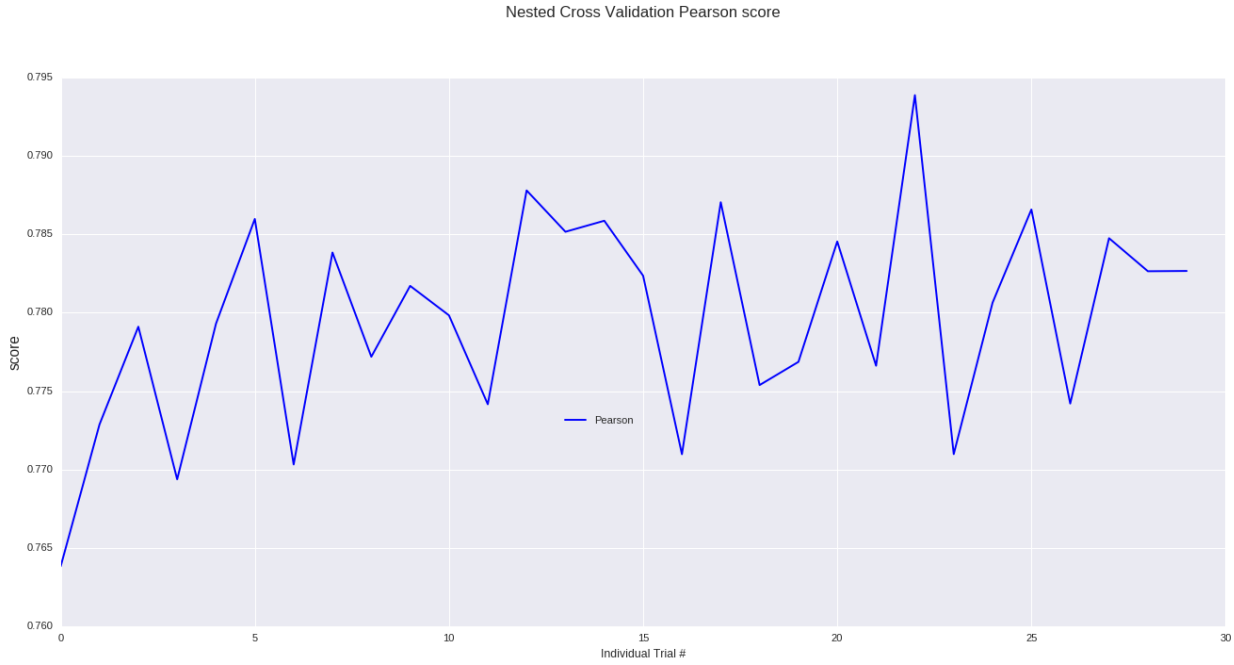


Figure 4: Pearson correlation score

domain specific knowledge embedded inside it and one which has no feature engineering.

One other way to tackle the problem is from bottom up: by modelling compositions of constituents of the language on various levels of abstractions: characters, words, sentences.

Although vector embeddings seem like a natural way to capture meaning on a word level, capturing composition of words within a sentence is a more difficult problem. Summation as an aggregation function doesn't effectively model semantic composition within a sentence. It is also invariant to word order within a sentence, while the meaning isn't. For example, changing the position of a negation within a sentence changes the meaning, but not the sum.

All of those ideas and problems are a motivation for applying a deep neural network on this task.

Table 3: Sample network output.

Sample output of the LSTM
Little Joe Otter had had no honour pointing to help him so that they heard of his own fish. And it was for the beginning of the Little Fir Tree and scrambled to the sheep. Come, let us go and let the right out of this tree, and the LORD had chanced to the next tabernacce of Mischians, when the king was carried the short of the children of the Green.

In this paper, one specific instance of such a model is explored: character-level language modelling with deep recurrent neural networks.

3.4.2. Recurrent Networks

The idea is to apply unsupervised learning with recurrent neural networks to learn a useful model of the language, which could then be used on the semantic similarity problem. In other words, we trained the model on a large corpus of text and used transfer learning to use the same trained network on STS.

We trained a Long short-term memory network (Hochreiter and Schmidhuber, 1997) on a task of predicting the next character in the complete NLTK Gutenberg corpus. We used a two-layer LSTM cell with 512 size of memory and cross entropy loss on the softmax outputs. 25% of the NLTK corpus was used as test data. Early stopping was used as an implicit method of regularizing. It is possible to generate arbitrary text with such a trained LSTM network by sampling from the predicted character probability distribution and feeding back that sample to the network. One such sample on a trained network is shown in the 3 and it helps show, subjectively, the level of comprehension LSTM has of the natural language.

Since the network learns an internal representation of the natural language, it is possible to extract the final hidden state of the network and use it as an embedding of the sentence that was fed in as the input. To put it in perspective, LSTM here serves as an aggregation function for the one-hot embeddings of the characters. It has the useful property of not being invariant to the position of characters. The final state of first sentence is then compared to the final state of the second sentence using the cosine similarity. In the case of a deep network, like the one we used, only the last layer's final state was used. Figure 5 shows the joint plot of LSTM network.

3.5. All Results

Table 4 shows the performance of all tested models. SVM greatly outperforms all other models.

Table 4: Model comparison.

Model	Pearson coefficient
Jaccard similarity	0.73
Cosine baseline	0.73
LSTM	0.61
SVM	0.78

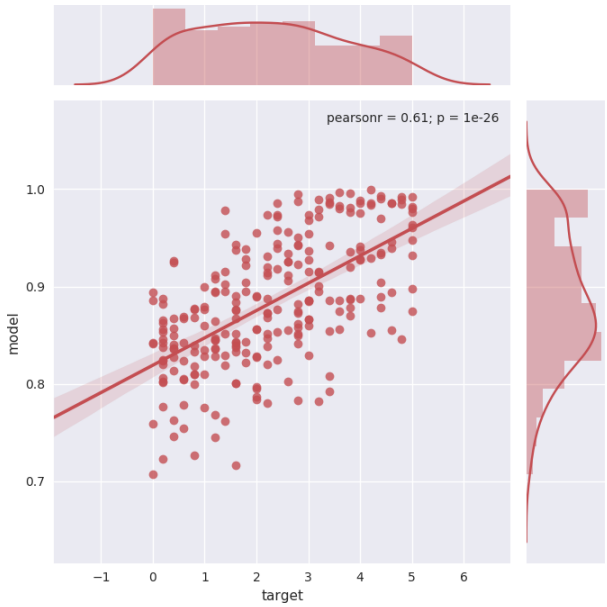


Figure 5: Depiction of assigned similarity measure by LSTM language model

4. Conclusion

As presented in this paper, we tried two way approach, both SVM and deep learning based one. As we're only students, dabbling with NLP giants, our results aren't considerably behind other papers and SemVal winner. Further improvement we see in bigger dataset, combining approaches with model ensables, and even combining various features with CRF, a techinque known for its good performance and resilience to linearly correlated features.

Deep neural network approach could be improved by modelling the language on various levels, not just character levels. Word embeddings and other network architectures could be utilized to gain additional performance gains.

Furthermore additional improvement could be made in bigger model search, both hyperparameters and basic model (e.g. Decision trees regression, logistic regression, etc.) using powerful Bayesian hyperoptimization frameworks such as Hyperopt (Bergstra et al., 2013)

References

Eneko Agirre, Mona Diab, Daniel Cer, and Aitor Gonzalez-Agirre. 2012. Semeval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-*

Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation, pages 385–393. Association for Computational Linguistics.

Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2016. Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. *Proceedings of SemEval*, pages 497–511.

Daniel Bär, Chris Biemann, Iryna Gurevych, and Torsten Zesch. 2012. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 435–440. Association for Computational Linguistics.

I. Beltagy, Katrin Erk, and Raymond Mooney. 2014. Probabilistic soft logic for semantic textual similarity. *Proceedings of Association for Computational Linguistics (ACL-14)*.

James Bergstra, Dan Yamins, and David D Cox. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pages 13–20.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJcAI*, volume 7, pages 1606–1611.

Lushan Han, Abhay Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. 2013. Umbc ebiquity-core: Semantic textual similarity systems. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics*, volume 1, pages 44–52.

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

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Frane Saric, Goran Glavas, Mladen Karan, Jan Snajder, and Bojana Dalbelo Basic. 2012. Takelab: Systems for measuring semantic text similarity. In *SemEval@NAACL-HLT*.

Md Arafat Sultan, Steven Bethard, and Tamara Sumner. 2014. Dls cu: Sentence similarity from word alignment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 241–246.

Md Arafat Sultan, Steven Bethard, and Tamara Sumner. 2015. Dls cu: Sentence similarity from word alignment and semantic vector composition. In *Proceedings of the 9th International Workshop on Semantic Evaluation*, pages 148–153.

Stylistic Context Clustering for Token-Level Author Diarization

Ivan Grubišić, Milan Pavlović

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ivan.grubisic, milan.pavlovic}@fer.hr

Abstract

In this paper we present an approach to tackle the PAN 2016 author diarization and intrinsic plagiarism detection problem. We propose a method for unsupervised text diarization based on clustering of tokens using contextual style features obtained with a sliding window. There are two variants of our approach - one that uses different features based on the document being analyzed, and one that uses a fixed feature space and tries to adapt it for clustering with a trainable feature transformation. We have evaluated our approach on the available datasets from PAN 2016.

1. Introduction

In this paper we will focus on the author diarization task proposed on the PAN 2016 competition.[1] The aim of this task is to decompose a document into its authorial parts, i.e., to split a text into segments and assign an author to every segment (Koppel et al., 2011; Aldebei et al., 2015). This is one of the unsupervised variants of a well known authorship attribution problem since text samples of known authorship are not available (Rosso et al., 2016). As we will describe, in two out of three subtasks of this task only a correct number of authors for a given document is known.

The simplest variant of the authorship attribution problem is about finding the most likely author for a given document from a set of candidate authors whose authentic writing examples are available (Stamatatos, 2009b; Stein et al., 2011; Ding et al., 2016). This problem can be tackled with supervised machine learning techniques as a single-label multiclass text classification problem, where one class represents one author (Stamatatos, 2009b).

The authorship attribution problem is also known as authorship identification and it is a part of authorship analysis (Stamatatos, 2009b; Ding et al., 2016). Authorship analysis is a field of stylometry and studies information about the authorship of a document, based on features derived from that document (Layton et al., 2013). Moreover, stylometry analyses literary style with statistical methods (Stein et al., 2011).

Rosso et al. (2016) divided the PAN 2016 author diarization task into three subtasks. The first subtask is traditionally called intrinsic plagiarism detection (IPD). The goal of this task is to find plagiarized parts of a document in which at least 70% of text is written by main author and the rest by one or more other authors. The term *intrinsic* means that a decision whether a part of the document is plagiarized has to be made only by analysing the given document, without any comparisons with external sources. This was our main motivation for solving this and other subtasks. In the rest of the paper we refer to this subtask as Task *a*.

Other two subtasks are more related to the general task of author diarization. In the second subtask we need to segment a given document and group identified segments by author. In the rest of the paper we refer to the second subtask as a Task *b*. The third subtask differs from the second one in the fact that exact number of authors is unknown. In the rest of the paper we refer to the third subtask as a Task *c*.

Table 1: Basic characteristics of training sets. * represents that there is a true author and plagiarism segments which do not have to originate from a single author.

Task	Number of documents	Average length (in tokens)	(min, max) authors
Task *a*	71	1679	(2, 2)*
Task *b*	55	3767	(2, 10)
Task *c*	54	3298	(2, 10)

For each of the three subtasks a training set is publicly available.[1] Rosso et al. (2016) explain that they are collections of various documents which are part of Webis-TRC-12 dataset (Potthast et al., 2013). Every document in that dataset is constructed from texts of various search results (i.e., authors) for one of the 150 topics in total. By varying different parameters such as the number and proportion of the authors, places in a document where an author switch occurs (between words, sentences or paragraphs), three training and test sets were generated (Rosso et al., 2016). Test datasets are currently not publicly available and we could not use them for evaluation of our approach. Some basic characteristics of the training sets are shown in Table 1.

2. Related work

The basic assumption in authorship analysis is that texts of different authors are mutually separable because each author has a more or less unique writing style (Stamatatos, 2009b; Ding et al., 2016). Therefore, the most of related work tries to distinguish writing styles by finding better features and methods which writing style will be quantified and measured with.

Zu Eissen and Stein (2006) used the average sentence

[1] http://pan.webis.de/clef16/pan16-web/author-identification.html

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 30–36, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

length, part of speech tags, the average stop word number and the averaged word frequency class as input features for their linear discriminant analysis and support vector machine (SVM) models. Stamatatos (2009a) introduced a sliding window approach with character tri-grams as input features for a style change function whose peaks indicate positions in the document where style changes occur. This is similar to an outlier detection approach from Stein et al. (2011), but they applied a naive Bayes' algorithm. Rahman (2015) also used a sliding window and an SVM, but introduced new kinds of information theoretical features.

Koppel et al. (2011) used normalized cuts algorithm to obtain initial clusters of segments which were represented only by normalized counts of synonyms from Hebrew synsets. An SVM was then used to classify bag-of-words feature vectors of non-representative cluster samples. Brooke et al. (2013) concluded that a very good initial segmentation of text, at least in poems written by T. S. Elliot, is needed for good performance of their modified k-means algorithm in clustering of voices.

The works by Kuznetsov et al. (2016) and Sittar et al. (2016) were submitted on the PAN 2016 competition for three aforementioned tasks. Their approaches mostly focused on initial segmentation with the help of a style change function and clustering as the final step. To estimate the unknown number of authors in Task *c*, Kuznetsov et al. (2016) defined a cluster discrepancy measure which was than maximized, while Sittar et al. (2016) generated that number randomly.

The most of the described approaches operate on the level of longer text segments or sentences. Since the style change in our tasks can occur even between two tokens in the same sentence, we wanted our model to be able to work on the token level. We were also inspired by Brooke et al. (2013) who said that a more radical approach would not separate the described tasks in segmentation and clustering steps, but rather build authorial segments that would also form good clusters. Instead of clustering tokens directly, we decided to cluster their vectorized stylistic contexts because they obviously contain more valuable stylistic information than tokens alone.

3. Author diarization and intrinsic plagiarism detection

We define a document D as a finite sequence of tokens $(t_i)_{i=1}^{n}$, where n can differ among documents. Given a document, each of its tokens is unique and defined by its character sequence and position in the document. Therefore, a document can be equivalently represented by its set of tokens $\{t_i\}_{i=1}^{n}$.

For each document, there is a corresponding mapping to a sequence of labels $(a_i)_{i=1}^{n}$ that are representing groupings of tokens by authors. The labels a_i are indices of authors of the document. Each token t_i is assigned a document-level label $a_i \in \{1..c\}$ associating it to one of c authors. The exact value of the label is not important. It is only required that all tokens corresponding to the same author have the same label. Therefore, there are $m!$ equivalent such mappings given a document. In the case of intrinsic plagiarism detection, there are only 2 labels: 0 (the main author), and 1 (plagiarized text).

Equivalently, the codomain of the mapping can also be defined as a set of segmentations. A segmentation S is a minimal set of segments, where each segment s is a set of consecutive tokens $\{t_i\}_{i=i_1}^{i_2}$ where each token is associated with the same author label. For a segmentation to be valid, the segments must cover all terms in the document and not overlap.

The correct mapping of a documents to the corresponding segmentations will be denoted with σ. Let $\mathcal{D}$ be a dataset consisting of a finite set of pairs of documents and corresponding segmentations, i.e., $\mathcal{D} = \{(D_i, \sigma(D_i))\}_{i=1}^{N}$. The goal is to find the model $\hat{\sigma}$ that approximates the correct mapping σ and generalizes well.

3.1. Evaluation measures

For evaluation of intrinsic plagiarism detection, Potthast et al. (2010) define multiple measures for different aspects of a system's performance. The main measures are binary macro-averaged and micro-averaged precision (P), recall (R) and F_1-score. For evaluating author diarization, we use *BCubed* precision, recall and F_1-score described by Amigó et al. (2009), which are specialized for evaluation of clustering results. The same measures were used for evaluation on the PAN 2016 competition (Rosso et al., 2016).

Let l be a function that associates lengths in characters to segments. Specially, $l(\{\}) = 0$. For notational convenience, we also use l to denote the sum of lengths of all segments in a set of segments: $l(S) = \sum_{s \in S} l(s)$, where S is as set of segments. Given a document D, let $S_\mathrm{p} \subseteq \sigma(D)$ be a set of all true plagiarism segments of the document and $\hat{S}_\mathrm{p} \subseteq \hat{\sigma}(D)$ the segments predicted as plagiarism by the model. With $S_\mathrm{tp} = \bigcup_{(s,\hat{s}) \in S_\mathrm{p} \times \hat{S}_\mathrm{p}} l(s \cap \hat{s})$, the micro-averaged evaluation measures for intrinsic plagiarism detection are defined as follows:

$$P_\mu = \frac{l(\hat{S}_\mathrm{tp})}{l(\hat{S}_\mathrm{p})}, \tag{1}$$

$$R_\mu = \frac{l(\hat{S}_\mathrm{tp})}{l(S_\mathrm{p})}, \tag{2}$$

$$F_\mu = 2(P_\mu^{-1} + R_\mu^{-1})^{-1}. \tag{3}$$

The macro-average evaluation measures treat all plagiarism segments equally regardless of their lengths:

$$P_\mathrm{M} = \frac{1}{|\hat{S}_\mathrm{p}|} \sum_{\hat{s} \in \hat{S}_\mathrm{p}} \frac{\sum_{s \in S_\mathrm{p}} l(s \cap \hat{s})}{l(\hat{s})}, \tag{4}$$

$$R_\mathrm{M} = \frac{1}{|S_\mathrm{p}|} \sum_{\hat{s} \in S_\mathrm{p}} \frac{\sum_{s \in \hat{S}_\mathrm{p}} l(s \cap \hat{s})}{l(s)}, \tag{5}$$

$$F_\mathrm{M} = 2(P_\mathrm{M}^{-1} + R_\mathrm{M}^{-1})^{-1}. \tag{6}$$

In author diarization, document segments have to be clustered into c clusters, where c is the number of authors that may or may not be known to the system. We divide the segments from the true segmentation S and the predicted segmentation $\hat{S}$ each into sets of segments S_i, $i \in \{1..c\}$, and $\hat{S}_j$, $j \in \{1..\hat{c}\}$, where c is the true number of authors,

and $\hat{c}$ the predicted number of authors. We use the following *BCubed* measures for evaluation:

$$P_{\mathrm{B}^3} = \sum_{i=1}^{c} \frac{1}{l(S_i)} \sum_{j=1}^{\hat{c}} \sum_{s \in S_i} \sum_{\hat{s} \in \hat{S}_j} l(s \cap \hat{s})^2, \quad (7)$$

$$R_{\mathrm{B}^3} = \sum_{j=1}^{\hat{c}} \frac{1}{l(\hat{S}_j)} \sum_{i=1}^{c} \sum_{s \in S_i} \sum_{\hat{s} \in \hat{S}_j} l(s \cap \hat{s})^2, \quad (8)$$

$$F_{\mathrm{B}^3} = 2(P_{\mathrm{B}^3}^{-1} + R_{\mathrm{B}^3}^{-1})^{-1}. \quad (9)$$

4. The proposed approach

Our approach can generally be described as a pipeline of three transformations: basic feature extraction f_{b}, feature transformation f_{t} and clustering f_{c}. The basic feature extractor denoted with f_{b} is used to extract stylistic features from the contexts of all tokens. If D is a document with n tokens, the basic feature extractor outputs a sequence of n n_{b}-dimensional feature vectors representing the contexts of tokens. The next step in the pipeline is the feature transformation f_{t} that maps the basic features to a n_{t}-dimensional space that they can be better clustered in. The final step in the pipeline is clustering denoted with f_{c}. The clustering algorithm implicitly clusters tokens because it actually clusters their stylistic contexts, each cluster representing an author. Depending on the task, the clustering algorithm can either be given a known number of authors, or try to predict it.

The following steps are done in predicting the segmentation: (1) raw text is tokenized giving a sequence of tokens D, (2) for all tokens, features are extracted from their contexts, giving a sequence of feature vectors $\phi_{\mathrm{t}} = (f_{\mathrm{t}} \circ f_{\mathrm{b}})(D)$, (3) the tokens are clustered based on ϕ_{t}, giving a sequence of author labels $(a_i)_{i=1}^{n}$, where n is the number of tokens in D, and (4) a segmentation $\hat{S} = f_{\mathrm{c}}(\phi_{\mathrm{t}})$ is generated based on the obtained clusters.

We develop two variants of our model which we will refer to as Model A and Model B. They are mainly differentiated by the fitting of the basic feature extractor, feature transformation learning and in the feature transformation. Model B utilizes a trainable feature transformation that requires a set of basic features with a fixed dimension. Therefore, the basic feature extractor must be fitted on the whole training set. Conversely, Model A's basic feature transformation can be performed specifically allowing for use of less more important features, but it can not use a feature transformation trainable on multiple documents. In both approaches basic features are extracted from a sliding window which moves from token to token and includes the context of each token.

Tokenization. As a preprocessing step, we tokenize each document using NLTK[2] by Bird et al. (2009) to obtain a sequence of tokens D. We also perform part-of-speech tagging with the same tool, to speed up basic feature extraction which we describe below. We did not use other preprocessing techniques such as lemmatization, stemming and stop word removal because they would take away a lot of stylometric data from text (Stamatatos (2009b). The final output from the tokenization step was a finite sequence $\{(t_i, o_i, l(t_i), POS_i)\}_{i=1}^{n}$ where o_i is the offset of token t_i, $l(t_i)$ its length in characters and POS_i its POS tag.

Basic feature extraction. We define the context of a token t_i as a set of tokens $\{t_k\}_{k=i-cs}^{i+cs} \setminus \{t_i\}$ where cs is context size. Based on previous work, we extract the most useful stylometric features from each context. The features considered in our models are:

- *Character tri-grams*. Frequencies of n-grams on character level have been very useful in quantifying the writing style (Stamatatos, 2009a). They are able to capture lexical and contextual information, use of punctuation and errors which can be an author's "fingerprint". This feature is also tolerant to noise. Based on work by Stamatatos (2009b) and Rahman (2015), we choose $n = 3$. Maximal dimension of this feature vector was set to 200.

- *Stop words*. According to Stamatatos (2009b), these are the most common used topic-independent words in text, such as articles, prepositions, pronouns and others. They are used unconsciously and found to be one of the most discriminative features in authorship attribution since they represent pure stylistic choices of authors' (Burrows, 1987; Argamon and Levitan, 2005). We used frequencies of 156 English stop words available in NLTK.

- *Special characters*. We used counts of all character sequences which satisfied the following regular expression: `[^\\w]{1,4}`. Although character n-grams can catch the use of those character sequences, we wanted to have a distinct feature for that purpose since Koppel et al. (2009) mentioned that authors can have different punctuation habits.

- *POS tag counts*. This is syntactic feature which Koppel et al. (2009) and Stamatatos (2009b) also identified as a discriminative one in authorship analysis and it was used by Kuznetsov et al. (2016). We used all 12 tags from the universal tag set available in NLTK.

- *Average token length*. Used by Kuznetsov et al. (2016), Sittar et al. (2016), Brooke and Hirst (2012) and Stein et al. (2011). Koppel et al. (2009) characterized this feature as a complexity measure.

- *Bag of Words*. Bag of words text representation more captures content, rather than style (Stamatatos, 2009b). We include this feature because it boosted performance in our initial testing. Counts of at most 100 unigrams are used.

- *Type-token ratio*[3]. This feature is the ratio of vocabulary size and total number of tokens of the text (Stamatatos, 2009b).

[2] `http://www.nltk.org`

[3] We wanted to use this feature to measure the vocabulary richness, but after we have evaluated performance, we realized that there was a bug in our implementation. The inpact on the results is unknown to us.

Feature transformation. As its feature transformation f_t, Model A uses feature scaling to zero mean and unit variance for each document separately. Model B uses a trainable transformation which requires a basic feature space of fixed dimension. Let $(\boldsymbol{b}_i)_{i=1}^n = f_\mathrm{b}(D)$ be a sequence of basic feature vectors and $(\boldsymbol{b}'_i)_{i=1}^n$ the corresponding sequence of potentially preprocessed basic feature vectors with elements from $\mathbb{R}^{n'_\mathrm{b}}$. Let $(a_i)_{i=1}^n$ be the sequence of true author labels with elements from $\{1..c\}$. We want to maximize the *clusterability* of the feature vectors obtained by the feature transformation $T : \mathbb{R}^{n'_\mathrm{b}} \to \mathbb{R}^{n_\mathrm{t}}$ with trainable parameters $\boldsymbol{\theta}_T$, i.e., we want to maximize segregation and compactness of groups of transformed feature vectors grouped by their target author label. Let $(\boldsymbol{t}_i)_{i=1}^n = (T(\boldsymbol{b}'_i))_{i=1}^n = f_\mathrm{t}((\boldsymbol{b}_i)_{i=1}^n)$ be the sequence of transformed feature vectors with elements from $\mathbb{R}^{n_\mathrm{t}}$. In order to optimize the transformation T, we define the following loss:

$$L = \alpha L_\mathrm{c} + (1 - \alpha) L_\mathrm{s} \tag{10}$$

with α chosen to be 0.5. Here L_c is the *compactness loss* and L_s the *segregation loss*. L_c is proportional to the average variance of groups, and L_s penalizes centroids being too close to each other. Let N_a represent the number of tokens associated with author a, i.e., $N_a = \sum_{i=1}^n [a_i = a]$, where, $[\varphi]$ evaluates to 1 if φ is true and 0 otherwise. Let $\boldsymbol{\mu}_a$ represent the current centroid of the transformed feature vectors associated with the author label a:

$$\boldsymbol{\mu}_a = \frac{1}{N_a} \sum_{i=1}^{n} T(\boldsymbol{b}'_i)[a_i = a]. \tag{11}$$

We define the components of the loss:

$$L_\mathrm{c} = \sum_{a=0}^{c} \frac{1}{N_a} \sum_{i=1}^{n} \|T(\boldsymbol{b}'_i) - \boldsymbol{\mu}_a\|^2 [a_i = a], \tag{12}$$

$$L_\mathrm{s} = \frac{2}{c} \sum_{a=0}^{c} \sum_{b=a+1}^{c} \|\boldsymbol{\mu}_a - \boldsymbol{\mu}_b\|^{-2}. \tag{13}$$

The *compactness loss* L_c is a weighted sum of within-group variances. The *segregation loss* L_s is proportional to the sum of magnitudes (not magnitude of the sum) of forces between c equally charged particles each located at one of the centroids. By minimizing the average loss (the error) across all documents in the training set with respect to the parameters of the transformation $\boldsymbol{\theta}_T$ we hope to benefit the clustering algorithm. Instead of squared L^2 distance, some other distance measure may be used. Also, the way of combining the two losses was somewhat arbitrarily chosen as well as the segregation loss. For the transformation T we have tried a nonlinear transformations represented as neural networks with one or more hidden layers. We did not explore nonlinear transformations much because they were slower and did not give as good results as a linear transformation or an elementwise linear transformation:

$$T_1(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}, \tag{14}$$

$$T_2(\boldsymbol{x}) = \boldsymbol{w} \odot \boldsymbol{x}. \tag{15}$$

Here $\boldsymbol{W} \in \mathbb{R}^{n'_\mathrm{b} \times n_\mathrm{t}}$ and $\boldsymbol{w} \in \mathbb{R}^{n'_\mathrm{b}}$. Before applying the transformations, basic feature vectors are preprocessed by concatanating each with the vector of its squared elements:

$$\boldsymbol{b'_i} = (\boldsymbol{b_i}^\mathsf{T}, (\boldsymbol{b_i} \odot \boldsymbol{b_i})^\mathsf{T})^\mathsf{T}, \quad i \in \{1..n\}. \tag{16}$$

The transformation is implemented using TensorFlow.[4] Parameters are randomly initialized with values from a truncated normal distribution with standard deviation 0.1. It is trained on-line (one document per optimization step) using RMSProp (Tieleman and Hinton, 2012) with default parameters and learning rate 5×10^{-4}.

Clustering. The final step in our approach is clustering of obtained feature vectors which represent stylistic contexts of tokens. The number of clusters is equal to the number of authors in tasks *a* and *b*, but in Task *c* it is unknown. We tested several clustering algorithms available in scikit-learn[5] Python toolkit by Pedregosa et al. (2011) in all tasks. In Model A for tasks *a* and *b* we propose hierarchical agglomerative clustering, and for Task *c* DBSCAN algorithm which estimates the number of clusters automatically. In Model B we use k-means for clustering in all tasks, and tried predicting the number of clusters based on the elbow rule. This is described in more detail in section 5.

5. Experimental results

For experiments, we have used the three PAN 2016 training sets. The test sets used in the competition have not been made publicly available. Therefore, our evaluation results might be only moderately comparable to the results of other systems.

In addition to the 2 models from the teams that took part in solving the author diarization problem on the PAN 2016 competition, we defined 3 baselines for evaluation purposes. The first baseline, Single-author dummy baseline, always assigns the whole document a single author label, or, in the case of intrinsic plagiarism detection, it labels the whole document as plagiarism. The second baseline, Stochastic dummy baseline, learns the average total lengths of authorial parts over all documents ranked by the length of their parts. For prediction, by modelling a Markov chain, it generates segmentations with similar label frequencies and predefined expected lengths.

In the third baseline, Simple baseline, we use default k-means clustering of vectorized document's sentences. Average token length, sentence length, universal POS tag counts and bag of at most 100 words (including stop words) serve as features. If the number of authors is unknown, it is randomly generated from a uniform distribution $(0, 0.0003 \times l(D))$ and increased by 1 if the result is 0.

We performed initial testing of usefulness of all features except for the accidentally wrongly implemented type-token ratio which was added afterwards. For testing purposes we used the implementation of the k-means algorithm from scikit-learn with the default settings and all documents from the dataset for Task *b*. As we were adding feature by feature, BCubed precision was becoming higher so we decided to use all features for Model A in all tasks. We do not

[4] `https://www.tensorflow.org`

[5] `http://scikit-learn.org`

know the exact impact of the wrongly implemented type-token ratio feature. Instead of the context's vocabulary size, we divided the size of document's vocabulary with the context size so this feature had the same value in all vectors.

5.1. Experimental setup

In every task we used the same tokenization procedure as described in section 4. Model A covers all three tasks and has several hyperparameters for each of them: context size, clustering algorithm and clustering algorithm's hyperparameters. Model B uses a smaller set of features and trainable feature transformations with most of the hyperparameters equal among the tasks. All the source code for our experiments is available in our repository.[6]

Model A. Sliding widow context size was determined manually for tasks *a* and *b*, in a similar way as feature selection. We used a whole dataset and default implementation of k-means from scikit-learn and tested different values. The best result for Task *a* was $cs_{\mathrm{a}} = 16$ and for Task *b* $cs_{\mathrm{b}} = 140$. For Task *c*, we left $cs_{\mathrm{c}} = 140$ as for Task *b*.

For clustering algorithm selection we used manual search over several algorithms. We split the whole dataset into a training and a test set. For each algorithm we were examining, a grid search of its hyperparameters was performed on the training set. We selected an algorithm and hyperparameters that achieved the best performance on the training set (since this is an unsupervised approach), and reported the final performance on the test set. We did not include the k-means algorithm in clustering model selection since we used it for context size estimation.

In Task *a*, we used 50% of the dataset as a training set, and the other half as a test set. Since this task is characterized as an outlier detection task, we tested an isolation forest algorithm, but achieved poor results. Other algorithm which we evaluated was hierarchical agglomerative clustering which achieved the best macro F_1 (described in subsection 3.1.) with cosine affinity and average linkage on the train set. We report its result on the test set.

For Task *b* we split the dataset in two halves as well. Grid search for the best hyperparameters was performed for agglomerative clustering. The best performance on the training set was obtained with Euclidean affinity and average linkage. For Task *c* we should have somehow determined the number of clusters. Due to a lack of time we decided to test algorithms which can determine this number automatically, i.e., this is not a hyperparameter which should be known in advance. Since we noticed that grid searches take a lot of time, we reduced a training set from 50% to 10% and repeated all experiments. We tested DBSCAN and affinity propagation algorithms.

The best results on Task *c* were obtained with DBSCAN. We noticed that the number of clusters estimated with the best hyerparameters (`eps=100, min_samples=71, metric="manhattan"`) was in almost every case greater than a real one, and solid number of examples were not even clustered because they were not close enough to any core sample to be a part of a cluster. In all grid searches we used the BCubed F_1-score and only DBSCAN achieved acceptable performance. Our opinion is that this was a consequence of a larger number of smaller clusters which had a positive influence on BCubed precision and therefore on overall performance.

Since the model with the best achieved Bcubed F_1-score (0.61) is not of practical use in real life applications because the number of clusters is always wrongly estimated, we decided to repeat the grid search in Task *c* for DBSCAN, but this time we used the Calinski-Harabaz index defined by Caliński and Harabasz (1974) as evaluation measure since it does not require the real number of clusters for clustering evaluation. Finally, from all results on training set we chose (`eps=0.23, min_samples=64, metric="cosine"`) for the final hyperparameters. This was not the best combination, but it produced relatively acceptable amount of noisy (non-clustered) elements per document, and performed quite fast but again did not estimate the correct number of clusters in most cases. If some samples were not clustered, we assigned them the majority label from neighborhood (context) of size 10 (5 on each side), or the majority label from all samples if samples in the context were not clustered.

Model B. For Model B, we have decided to use a smaller set of basic features: character tri-gram counts (for 120 trigrams), stop word counts, average token length and bag-of-words (for 120 words). The length of the sliding window is set to 120 tokens. Model B uses k-means clustering. We have chosen two trainable feature transformations: linear and elementwise linear (weighting). For both of the transformations 5×10^{-4} was chosen for the learning rate. All the hyperparameters are selected manually by experimenting on the development dataset, which is approximately 70% of the available dataset. 70% of the development set was used for fitting basic feature extractors and learning the transformation. Considering the results on the validation part of the development set, not very large numbers of iterations seem to be more beneficial. A number of 40 iterations was chosen for the linear transformation, and 20 for the elementwise linear transformation. For the output dimension of the linear transformation we have chosen $n_{\mathrm{t}} = 40$, though not much different results seem to be achievable using smaller numbers like 20, making training and clustering faster.

For Task *c* we have implemented a simple wrapper for the k-means algorithm that tries to predict the correct number of authors. It computes the k-means error for each number k in the specified range for the number of clusters. Based on ratios of relative improvements in the score between consecutive k-s, it chooses the k giving the best ratio modulated with a modulation function dependent on k, simply k^{α} was chosen, where α is selected using linear search. Being limited in time, we did not put much effort into optimizing the selection of the best number of clusters. Moreover, even if knowing the number of clusters in advance, it seems that even the transformed features are not satisfactory for correct clustering. We notice that using the correct numbers of authors compared to using a fixed $k = 2$ results in relative decrease in BCubed precision, recall, and F_1-score by factors of 0.95, 0.64 and 0.79 respectively.

[6] `https://github.com/Coolcumber/inpladesys/`

Table 2: Results on the intrinsic plagiarism detection task. The standard errors both for F_μ and F_M are for Model A within ± 0.04 and for Model B within ± 0.06. Results of models denoted with * were obtained on the PAN 2016 test set which was not available to us.

Method	P_μ	R_μ	F_μ	P_M	R_M	F_M
Single-author dummy BL	0.13	**1.00**	0.22	0.13	**1.00**	0.22
Stochastic dummy BL	0.11	0.09	0.09	0.08	0.09	0.06
Simple BL	0.13	0.24	0.14	0.13	0.24	0.13
Kuznetsov et al. (2016)*	**0.29**	0.19	0.22	**0.28**	0.15	0.17
Sittar et al. (2016)*	0.14	0.07	0.08	0.14	0.10	0.10
Model A	0.18	0.43	**0.25**	0.18	0.43	**0.24**
Model B (linear)	0.21	0.35	0.23	0.21	0.34	0.22
Model B (weighting)	0.24	0.36	0.23	0.24	0.34	0.23

Table 3: Results on the author diarization task for known (Task *b*) and unknown (Task *c*) numbers of authors. The standard errors for both tasks are for Model A within ± 0.02 and for Model B within ± 0.05. Results of models denoted with * were obtained on the PAN 2016 test set which was not available to us.

Method	Task *b*			Task *c*		
	P_{B^3}	R_{B^3}	F_{B^3}	P_{B^3}	R_{B^3}	F_{B^3}
Single-author dummy BL	0.33	**1.00**	0.47	0.37	**1.00**	0.52
Stochastic dummy BL	0.35	0.59	0.41	0.39	0.59	0.45
Simple BL	0.43	0.50	0.45	0.45	0.54	0.45
Kuznetsov et al. (2016)*	**0.64**	0.46	0.52	0.64	0.42	0.48
Sittar et al. (2016)*	0.28	0.47	0.32	0.31	0.47	0.35
Model A	0.61	0.65	**0.62**	**0.71**	0.50	0.54
Model B (linear)	0.48	0.45	0.46	0.56	0.82	**0.64**
Model B (weighting)	0.51	0.49	0.50	0.57	0.77	**0.64**

5.2. Evaluation results

We have evaluated our models using micro-averaged and macro-averaged precision, recall and F_1-score for intrinsic plagiarism detection and the analogous BCubed scores for author diarization, as defined in subsection 3.1. The results of our models are compared to the results of our baselines, as well as the results of the two teams that took part in the PAN 2016 competition for this problem. The results of the two teams have been evaluated on test sets that are not publicly available. Only the training sets from the competition have been available to us.

The results of evaluation for the intrinsic plagiarism problem are shown in Table 2. For author diarization with known and unknown numbers of authors, the results are shown in Table 3. It should be noted that the evaluated Model B uses a fixed number of clusters $k = 2$ for task *c* and that the results are actually not very good. By trying a more honest approach with predicting the number of clusters the BCubed F_1-score drops below 0.5. The results of Stohastic dummy baseline for all tasks are obtained over the 50 runs, as well as for Simple baseline in Task *c*.

For both Models A and B we calculated standard errors and wanted to obtain 95% t-based confidence intervals. Since the test sets are small, D'Agostino and Pearson's normality test with signficance level 0.05 was performed first. The Model A data in all tasks did not provide enough evidence to reject the null hypothesis that a sample comes from a normal distribution. In test of Model B's both F_1-scores, the data provided enough evidence to reject the null hypothesis.

Although the Model A achieved the highest micro and macro-averaged F_1-scores on the Task *a*, this task seemed to be very difficult. F_1-scores of Model A are only slightly above the Single-author dummy baseline and lower 95% limit for macro-F_1 is 0.15. Model A also achieved the best, but actually not very good performance on the Task *b* with $(0.57, 0.67)$ 95% confidence interval. Although Model B behaved reasonably well on the Task *c*, a better way of estimating the number of clusters is needed.

6. Conclusion

Our results, as well as the the results of others, suggest that author diarization and intrinsic plagiarism detection are very challenging problems. Our approach tries to solve the problem by clustering contextual writing style features extracted with a sliding window around each token.

We have presented some ideas that are not completely explored but, based on the data we have available, seem to

give comparable, or even better results than the other two approaches that tried to tackle the same problem. There is a lot of room for improvement of our approach, as well as in exploring other new ideas.

For further work, it would be good to search for more useful basic features, make a deeper examination of the feature transformation, try other clustering algorithms and a use more systematic approach for choosing hyperparameters. It would be also interesting to try the trainable feature transformation approach on other authorship analysis problems.

References

Khaled Aldebei, Xiangjian He, and Jie Yang. 2015. Unsupervised Decomposition of a Multi-Author Document Based on Naive-Bayesian Model. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers)*, pages 501–505.

Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. 2009. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information retrieval*, 12(4):461–486.

Shlomo Argamon and Shlomo Levitan. 2005. Measuring the usefulness of function words for authorship attribution. In *Proceedings of the 2005 ACH/ALLC Conference*.

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

Julian Brooke and Graeme Hirst. 2012. Paragraph clustering for intrinsic plagiarism detection using a stylistic vector space model with extrinsic features. In *CLEF (Online Working Notes/Labs/Workshop)*.

Julian Brooke, Graeme Hirst, and Adam Hammond. 2013. Clustering voices in the Waste Land. In *Proceedings of the 2nd Workshop on Computational Literature for Literature (CLFL'13), Atlanta*.

John F Burrows. 1987. Word-patterns and story-shapes: The statistical analysis of narrative style. *Literary and linguistic Computing*, 2(2):61–70.

Tadeusz Caliński and Jerzy Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27.

Steven HH Ding, Benjamin Fung, Farkhund Iqbal, and William K Cheung. 2016. Learning stylometric representations for authorship analysis. *arXiv preprint arXiv:1606.01219*.

Moshe Koppel, Jonathan Schler, and Shlomo Argamon. 2009. Computational methods in authorship attribution. *Journal of the American Society for information Science and Technology*, 60(1):9–26.

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 Kuznetsov, Anastasia Motrenko, Rita Kuznetsova, and Vadim Strijov. 2016. Methods for intrinsic plagiarism detection and author diarization. *Working Notes Papers of the CLEF*.

Robert Layton, Paul Watters, and Richard Dazeley. 2013. Automated unsupervised authorship analysis using evidence accumulation clustering. *Natural Language Engineering*, 19(01):95–120.

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.

Martin Potthast, Benno Stein, Alberto Barrón-Cedeño, and Paolo Rosso. 2010. An evaluation framework for plagiarism detection. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 997–1005. Association for Computational Linguistics.

Martin Potthast, Matthias Hagen, Michael Völske, and Benno Stein. 2013. Crowdsourcing interaction logs to understand text reuse from the web. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 1212–1221. Association for Computational Linguistics.

Rashedur Rahman. 2015. Information Theoretical and Statistical Features for Intrinsic Plagiarism Detection. In *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 144.

Paolo Rosso, Francisco Rangel, Martin Potthast, Efstathios Stamatatos, Michael Tschuggnall, and Benno Stein. 2016. Overview of pan'16. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 332–350. Springer.

Abdul Sittar, Hafiz Rizwan Iqbal, and A. Nawab. 2016. Author Diarization Using Cluster-Distance Approach. *Working Notes Papers of the CLEF*.

Efstathios Stamatatos. 2009a. Intrinsic plagiarism detection using character n-gram profiles. In *3rd PAN Workshop. Uncovering Plagiarism, Authorship and Social Software Misuse*, pages 38–46.

Efstathios Stamatatos. 2009b. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology*, 60(3):538–556.

Benno Stein, Nedim Lipka, and Peter Prettenhofer. 2011. Intrinsic plagiarism analysis. *Language Resources and Evaluation*, 45(1):63–82.

T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

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

Author Profiling Based on Age and Gender Prediction Using Stylometric Features

Alen Hrga, Karlo Pavlović, Tin Široki

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{alen.hrga, karlo.pavlovic, tin.siroki}@fer.hr

Abstract

Author profiling is an approach of growing importance in a lot of applications, like marketing and security. If it is done properly, one could identify author of some text message based on writing characteristics of some person. Also, companies may be interested in knowing whether users like their products, the demographics of people who use and review their products, and so on. This paper describes our approach in tackling author profiling task on PAN@CLEF 2016 dataset. The goal of this author profiling task is identification of gender and age group of an unknown Twitter user. For this task, we trained our model using lexical, syntactic character-based features and vocabulary richness on English part of given dataset using two different classifiers one for age and one for gender classification.

1. Introduction

A lot of people today use social networks like Facebook or Twitter to share their experiences and express opinions and beliefs. They all generate huge amount of information and data that can be used for analysis and have characteristics that differs them from others. Each person has attitude toward using emoticons, hashtags, punctuation marks and so on. We can use their distinct style of writing and their characteristics to categorize them on gender and age group basis. This paper presents our approach of identifying users gender and age group based on their tweets given in PAN@CLEF 2016[1] dataset. We train two different classifiers one for age and one for gender classification with different features. For age classification we use lexical features, syntactic features and vocabulary richness (Ashraf et. al., 2016). With the extracted features a support vector classifier is trained. The gender classification is done by using simple term frequency-inverse document frequency (tf-idf) values as features for training linear support vector machine with stochastic gradient descent. The implementation, testing and evaluating of our model is done with Python 3 programming language with SKLEARN and NLTK libraries.

This paper is organized in sections: Section 2 describes related work and Section 3 describes description of dataset used to test and evaluation of our model. Section 4 describes our proposed approach, what we do in our research and conducted experiments. Section 5 explains results of the experiments and evaluation of our model and Section 6 describes conclusions we get from described work and approach and describes what we will do in the future.

2. Related Work

Previous studies have commonly used several different approaches to identify user groups. Some of them tackled author profiling task with stylometry-based approach (Ashraf et. al., 2016) or as classification problem using stacking technique (Agrawal et. al., 2016). In our paper we adopted the former approach for age classification. Stylometrics is the study of linguistic style, stylometric features try to capture authors style of writing. Authors styles are different but the assumption is that authors with similar age may have similar writing style, hence the style information could be a good indicator of ones age.

3. Dataset Description

We use PAN@CLEF dataset which contains English, Spanish and Dutch tweets, but our model is trained only with English part of this dataset. Documents are XML documents containing tweets of author (there is one author per file) and there is truth.txt file which contains correct gender and age group of every author. English datasets consists of tweets from 436 authors (218 female and 218 male authors, and even 182 authors from 35-49 age group). Age groups are 18-24, 25-34, 35-49, 50-64, 65-xx and number of documents belonging to each group is different.

4. Proposed Approach and Experiments

Our approach tackles the problem by training two different classifiers, one for gender classification and one for age classification. Each problem consists of three stages:

- Preprocessing data
- Feature extraction
- Classification

The preprocessing part is same for both classifiers, but the features and models are not, so the feature extraction and classification part is explained separately for age and gender.

4.1. Preprocessing Data

For the preprocessing part, we used Python 3 programming language and NLTK library for natural language processing. Firstly, we loaded truth.txt file into memory for evaluation use and then loaded all data from English part of PAN@CLEF 2016 author profiling dataset.

[1] https://www.dropbox.com/s/x7zsudnnthccokq/pan16-author-profiling-training-dataset-english-2016-04-25.zip?dl=0

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 37–39, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

Table 1: The number of users belonging to the group.

Male	Female	18-24	25-34	35-49	50-64	65-xx
218	218	28	140	182	80	6

While loading data points, we removed XML tags from each XML file (truth.txt was excluded) and processed it. Processing data point implied loading every tweet of current user and extracting only data part (CDATA section) from XML tag. After that, we removed HTML and XML tags inside of CDATA section (which included raw tweet text) and HTML symbols (like lg; or amp;). After all HTML and XML symbols were removed, we parsed tweet text to replace all mentions (@ symbols) with our __MENTION__ tag, all hashtags with our __HASHTAG__ tag, all emojis with our __EMOJI__ tag and all links with our __LINK__ tag. Extra whitespaces were removed and all tweets were tokenized in the sentence-level.

4.2. Feature Extraction and Age Classification

We used different sets of features for classify stage of our system.

We used average word count, average emojis count, average hashtags count, average mentions count, average links count and vocabulary richness (average number of different words). Averages were used because there are different number of tweets per user and per file.

We used character count, percentage of punctuation characters per file, character without spaces count, percentage of semicolons, digits, commas, letters, apostrophes and colons.

We used NLTK POS tagger to find syntactic-based features and extracted number of adjectives, number of nouns, number of adverbs, number of verbs, number of particles, number of symbols, number of determiners, number of pronouns, number of conjunction, number of propositions and number of particles.

After the features are extracted and the feature vector is generated we use a standard scaler from sklearn library[2]. Standard scaler removes the mean and scales to unit variance. The centering and scaling is done for every feature independently. The scaling is done to eliminate possible domination of features which have variance orders of magnitude greater than other features. For the classification task we train a support vector classifier from sklearn[3] with a radial basis function kernel. The C and Gamma parameters of the model are optimized by cross validation on the training set.

4.3. Feature Extraction and Gender Classification

As features for gender classification we used term frequency-inverse document frequency (tf-idf) values for unigrams. After extracting unigrams from users tweet collection, we used lemmatization to reduce number of unique unigrams. We used lemmatization over stemming as it produced marginally better results. For lemmatization was used NLTK WordNetLemmatizer. CountVectorizer[4] was used to create complete dictionary out of unigrams collection used for training data and to organize users unigrams in a matrix of token counts. In the following step TfidfTransformer[5] was used to create a normalized tf or tf-idf representation. As a result we were able to scale down the impact of tokens that occur very frequently and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

For classification we used SGDClassifier[6]. This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). Optimised parameters were alpha which is a constant that multiplies the regularization term and penalty which is added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both.

5. Results and Evaluation

To compare how our model is performing we calculated the majority class classification baseline. For gender classification the classes are equally distributed between male and female, so the baseline accuracy is 0.5. In age classification the most numerous class is between 35 to 49 years of age, counting 182 items. Given the total size of the dataset is 436 the age accuracy baseline is 182/436 = 0.41. Taking combined classes for age and gender the class with the most items is male from 35 to 49 years of age counting 91 items, so the combined accuracy baseline is 91/436 = 0.20. We evaluated our model by using nested cross validation with 7 outer folds and 3 inner folds. For the inner loop which is used to optimize models hyperparameters we used a sklearn implementation of grid search[7], which internally uses cross validation for every combination of parameters in the grid. For our final accuracy results we have taken average accuracy scores of 7 iterations of training on 6/7 parts of the dataset and testing on the left 1/7 part. The accuracy

[2] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[3] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.htmlsklearn.svm.SVC

[4] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[5] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

[6] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[7] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Table 2: Accuracy score for English dataset.

Gender	Age	Combined
0.74	0.44	0.30

is calculated separately for gender, age and combined. For combined accuracy a data point is taken to be classified correctly if both age and gender are correct. Table 2. shows our accuracy scores for the given English dataset. The test was done on a standard personal computer with a time of execution of 30 minutes.

6. Conclusion and Future Work

In this work we tackle the problem of author profiling by training two different classifiers, one for gender classification and one for age classification. The gender, age and combined classification accuracies were 0.74, 0.44 and 0.30. The age classification has been proven to be a much harder task than gender classification, what is expected because gender classification is binary classification and age classification is multi class classification problem. Perhaps the difficulty comes from the dataset which is contained of tweets. Tweets tend to be written in a similar style and our features probably did not represent the subtle differences in styles of differently aged authors in an appropriate manner. For future work we leave exploring and extracting different features for age classification.

References

Madhulika Agrawal and Teresa Gonçalves 2016. *Age and Gender Identification using Stacking for Classification*. Notebook for PAN at CLEF 2016.

Shaina Ashraf, Hafiz Rizwan Iqbal, Rao Muhammad Adeel Nawab 2016. *Cross-Genre Author Profile Prediction Using Stylometry-Based Approach*. Notebook for PAN at CLEF 2016.

Paolo Rosso, Francisco Rangel, Martin Potthast, Efstathios Stamatatos, Michael Tschuggnall and Benno Stein 2016. *New Challenges for Authorship Analysis: Cross-genre Profiling, Clustering, Diarization, and Obfuscation*. Notebook for PAN at CLEF 2016.

Sentiment Analysis of Tweets Using Semantic Reinforced Bag-of-Words Models

Mate Kokan, Luka Škugor

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{mate.kokan, luka.skugor}@fer.hr

Abstract

In this paper we tackle the subject of feature level sentiment analysis in Twitter. The main motivation behind this problem is gathering and analyzing public opinion on certain topic, which is very useful in various fields like journalism and business. For this task we built simple system that is based on bag-of-words model, where tweet is represented as a vector of features that resemble unique words. Some of these features were removed or modified based on their qualities. With this data we trained and evaluated our SVM classifier. At the end, we compared our results with today's most popular approaches that are used to solve this difficult problem.

1. Introduction

This project is based on SemEval-2017 Task 4,[1] and we drew some inspirations from different approaches that were used (Nakov et al., 2016). It is interesting to note that this task in particular is most popular one as it attracts the highest number of participants than any other task. The task itself can be divided into three subtasks (two of which come in two different flavors).

Fistly, we are given a tweet, and our system needs to classify its sentiment whether it is positive, neutral or negative. This task for the most part was focal point of our work, as it gives us general picture how well our system functions. Also, the dataset provided by the competition was the richest one compared to the datasets for other subtasks. Secondly, we are given a tweet and its topic, and we need to classify it on two point scale, whether it is positive or negative. Besides classification, we need to solve quantification problem as well. For the set of tweets that are about certain topic, we need to determine distribution of positive and negative ones for every topic. Finally, we are also given a tweet and its topic, but this time we need to classify it on a five-point scale, where besides positive, neutral and negative classification, there are also highly positive and highly negative classification. We had to find distribution of these classes for each topic for this task as well.

2. Related Work

Sentiment analysis is very popular task in natural language processing, as lot of different approaches have been attempted to tackle this difficult problem. The first paper in this field has proposed the solution (Turney, 2002) by using patterns of words to extract phrases from reviews. As of lately, it seems that deep alghorithms (Socher et al., 2013) have taken dominance according to 2016 competition (Nakov et al., 2016). It is also important to mention powerful word embedding models such as word2vec and GloVe (Pennington et al., 2014). On the contrast, we used simple bag-of-words model along with traditional SVM classifier. With this system we are trying to show how simple methods perform in this rather complex task.

[1] http://alt.qcri.org/semeval2017/task4/

3. Models

As the baseline for training and testing our model we decided to use bag-of-words for translating our tweets into vectors of occurences of each unique word. This means that before training our classiffier we had to do some processing. The first processing we did was to extract vocabulary from our training corpus. This was done by tokenizing tweets and then selecting previously unseen unique lemmatized tokens. We used this extracted set of words as the base for builing our bag-of-words vectors. We tried to modify some of those features based on the hypothesis that not all tokens have the same contribution in the final prediction. While some of them essentialy do not hold any sentiment, others (or rather their forms) may enhance sentiment. We increased importance of these words by giving them some extra occurence value (instead of normal 1) in the bag-of-words vector representing the tweet. It is important to note that while building our bag-of-words vectors we were also monitoring every word's occurence, i.e. how many times it was present in each of the classes (depending on task). With all of this in mind, we did following modifications:

- **Based on part-of-speech tag**. We realized that most of the tweet's sentiment is projected through adjectives and adverbs. As we feared that the selection would be too narrow, we also kept nouns and verbs, since words like *hate* and *love* (both as verb and noun) can be good guides for tweet's sentiment. Words that are not tagged as noun, verb, adjective or adverb are therefore discarded.

- **Superlatives and comparatives**. As our part-of-speech tagger recognizes these forms of adjectives and adverbs (at least one-word versions, others are explained in section 3.2.), we decided that they should have greater importance in recognizing sentiment than their normal forms. This is intuitive because when someone says that they are *the happiest*, it is generally a lot stronger sentiment than if they only say that they are *happy*.

- **Words in mostly uppercase and repeating characters**. Importance of these words were increased

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 40–43, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

as well because intuitively such words also provide stronger sentiment. One such example would be when someone says that he is *HAPPYYYY*, where he clearly typed word *happy* this way to strenghten its sentiment. These words were normalized by naive methods so that they can be compared to their regular forms.

- **Hashtags and emoticons**. Sentiment of the tweet is usually encapsulated within hashtag or emoticon if any of them are present. Therefore we brought up importance of these tokens. Emoticons with similiar emotions are grouped into one feature to slightly reduce redundancy of overall features.

Tokens with following qualities were removed:

- **Low occurence rate among train set**. Words like these are not kept in our dictionary (set of unique words). This happens alot as general vocabulary in Twitter is rich, especially since there are lot of misspelling and slang words. This saved us a lot of processing time.
- **Quoting**. Given that we classify sentiment of opinion holder, all words that are surrounded in quotes should not contribute to the sentiment of the tweet, but rather the opinion of the quoted.
- **Tokens that have no alphabet letter**. These include numbers, punctuation marks (excluding question mark) and other random tokens that are not emoticons.

It is important to note that tweets from both train and test set were translated into vectors with this method, and the process does not depend on tweet's classification. Finally, before training our classifier, we removed tokens that do not strongly suggest toward one classification. For example, if a word *dog* appeared in 50 positive, 45 neutral and 5 negative tweets, we removed it as a feature because it is not clear if such word holds positive or neutral sentiment. We didn't consult SentiWordNet (Esuli and Sebastiani, 2006) on top of that as the dataset itself was filtered based on the SentiWordNet's results (Nakov et al., 2016). We weren't searching for named entities either for the same reason. As we felt that these modifications weren't enough, we also did some additional experiments on modifying features which yielded some interesting results.

3.1. Swearing

One would argue that a presence of a curse word would be a good sign that the sentiment of the tweet is more toward the negative side. We put that theory to test. We downloaded list of curse words off the site[2]. Then we had to observe how the presence of these types of words affect the sentiment. The idea was that if swears really do bring up negative sentiment, we should also increase their importance in the building of our vectors. From all the data we extracted tweets that have curse words in them. Then we trained linear SVM where elements of input vectors are number of occurences of that curse word. Then on our trained model we observed weights for each classification: positive, neutral and negative. The results were quite interesting. While there are couple of very mean words that definitely set tweet's sentiment as negative, there are almost no difference in the usage of most used ones when it comes to the sentiment. Often than not, these words actually enhance the sentiment, or how we like to call "amplify" it. This inspired us to implement little addition to our system that is explained in subsection 3.2.. We concluded that people today do not neccesarily use swears as offensive remark, but to make their opinion stronger.

[2]http://www.bannedwordlist.com/

3.2. Sentiment Amplifiers and Negations

Sentiment amplifiers are words that are adjectives or adverbs, and while they don't carry sentiment by themselves, they actually amplify sentiment of next adjective or adverb. Most intuitive example are comparatives and superlatives. While short adjectives have natural ones, for example: *happy*, *happier*, *happiest*, a lot of them have addition of more and most: *difficult*, *more difficult*, *most difficult*. Therefore, the presence of these words bring up the sentiment of the adjective or adverb they refer to. Also we included negations such as *no* and *not* (also *n't* token as it is result of tokenizing words like *shouldn't*). We implemented this as a dictionary where key is amplifier word and value represents how much it "amplifies" next word. Next snippet of code is part of method that translates tweet to bag-of-words vector.

foreach $word$ ***in*** $tweet$ **do**
  if $word \in amps$ **then**
    if $next\_word \in \{adjective, adverb\}$ **then**
      $BOW[next\_word] =$
       $BOW[next\_word] + amps[word]$;
      skip $next\_word$;
end

where *BOW* is our bag-of-words vector used as dictionary. Value of word *more* is 2, *most* 3, *not* -1 etc. If an adjective or adverb have negation prior to them, this approach by using negative occurence values is practically telling us that this word does not really "belong" in the annotated class of the tweet. Also we took into account amplifiers that are uppercase so it brings up some extra sentiment (for example *NOT* -1.5, *MOST* 3.5). We didn't really optimize these too much as there are many of them. We put the values to whatever made most sense, and changed them on the go if there would be a good argument to do so.

3.3. Classifier

As for our classifier, we first went with naive Bayes, but later switched to linear SVM as it gave better results. We did 10-fold cross-validation with both classifiers and then compared their F_1 scores using quantile-quantile (q-q) plot to determine whether they follow same distribution. Only then we could compare classifiers using paired t-test. SVM classifier turned out to be statistically better than the naive Bayes classifier at the $p < 0.01$ level.

After choosing linear SVM as our main classifier, we had to determine its best hyperparameter C. We computed both

recall and F_1 score in a nested 10-fold cross-validation to see for which hyperparameter C these results are the best. Candidates for best C were numbers $2^{-5}, 2^{-4}, \ldots, 2^{8}$. Results of cross-validation are shown on Figure 1 for recall and F_1 score. We determined that the hypeparameter C with which our model scores most consistency is 2^{-4}.

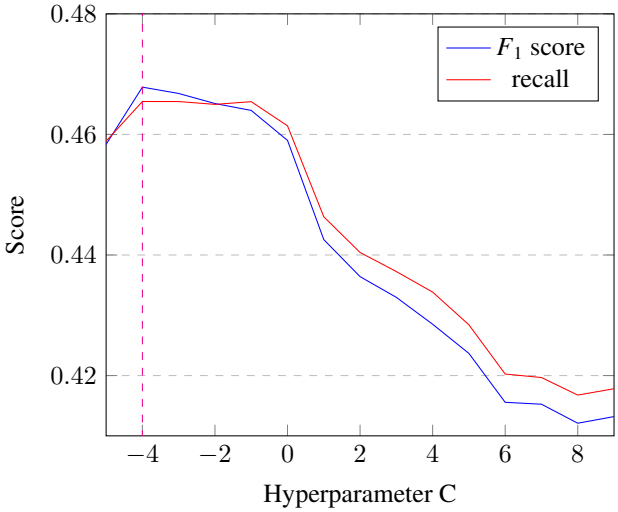


Figure 1: Results of our nested 10-fold cross validation. Training set used is of 2016 competition.

As we focused most of our time building our system for the first task, we really didn't have much to change it for the other tasks, as they are practicaly derived from the first task. Any actual improvement in tweaking our system for them was negligible. This was the case for all of the systems that competeted across all of the tasks as well.

4. Dataset

One privilege of having popular task such as sentiment analysis is vastness of annotated data. We used dataset that was provided by SemEval-2017 task 4 competition. Tweets were annotated using Amazon's Mechanical Turk and CrowdFlower. For our first task we were provided by dataset from previous four runs. We mainly focused on last year's dataset as it was most revelant for showing us how it would compete with today's most popular systems, but we also sometimes used older datasets to get wider picture of how our system functions. The 2016 version of training set consisted of 6000 total tweets, out of which 3094 were positive, 863 were negative and 2043 were neutral. The test set consisted of 20633 tweets in total, 7059 of which were positive, 3231 were negative and 2043 were neutral. Datasets for other two tasks are of smaller sizes, where in one tweets are annotated on two-point scale, while in other they are annotated on five-point scale.

5. Evaluation and Experiments

Just to see how our system would fare in the last competition, we used evaluation metrics prescribed in the competition's guidelines. Ultimately, our system scored better than all of the competition baselines.

For the first task prescribed metrics for 2016 competition was macroaveraged F_1 score over positive and negative class. For this year this was changed to macroaveraged recall. We trained our model on the whole train set, and tested it on test set of 2016. Results are shown on Table 1.

Table 1: Score for the first task. F_1 score and recall are computed over positive and negative classes.

System	F_1 score	Recall
Best system	0.63	0.67
Our score	**0.44**	**0.46**
Baseline system	0.25	0.33

Table 2: Full scores of our system for the first task.

Class	Precision	Recall	F_1 score
negative	0.44	0.23	0.30
neutral	0.60	0.52	0.56
positive	0.49	0.70	0.58

Even though we scored better than all of baselines for both measures, our system is still a lot behind best systems of the 2016 competition. As a comparison, most popular systems in the competition were convolutional neural networks and other deep learning algorithms. This is expected as deep learning is getting a lot of attention lately in solving these types of tasks. Complete results of our experiment are shown on Table 2.

As we can see, our downfall is mainly negative class, primarily its recall. This is generally considered as the main problem in this task. As we went through our processed sentences in test set, we concluded that in lot of situations negative sentiments can not overcome dominant neutral and positive sentiments consisted in mostly nouns. This is due to the fact that negative tweets are greatly outnumbered by positive and neutral tweets in train set.

In our second task we were given tweet along with its topic, and our system had to classify tweet on a two-point scale. We evaluated our model on 2016 dataset using *macroaveraged recall* over positive and negative class. While having almost 100% recall of positive classification, recall of negative classification was abysmal and it dropped our performace overall. This was expected though, as the discrepancy between number of positive and negative tweets is also present in this task. In addition, we were also to evaluate distribution of our prediction when given a set of tweets about certain topic. We used *Kullback-Leibler divergence* as a measure for this case. The real score is averaged over scores for each topic. As explained in competition guidelines (Nakov et al., 2016), we did some smoothing, as predicted distribution (denominator) can sometimes be zero. Our scores are shown on Table 3.

Finally for our third task we were given a tweet and its topic that we had to classify on a five-point scale. For the evaluation we used *macroaveraged mean absolute error*. The reason why we didn't use recall or F_1 score is that

Table 3: Results of the second task. Recall is computed over positive and negative classes. *KLD* is error score, lower it is the better.

System	Recall	*KLD*
Best system	0.79	0.03
Our score	**0.56**	**0.33**
Baseline system	0.50	0.88

Table 4: Results of the third task. Both *MAE* and *EMD* are error scores, lower they are the better.

System	*MAE*	*EMD*
Best system	0.719	0.243
Our score	**1.113**	**0.467**
Baseline system	1.200	0.734

we wanted to penalize heavier mistakes. For example, it is a bigger mistake if our model classifies tweet as highly positive, whereas it is actually highly negative than if it is just positive. For quantification problem, we used *Earth Mover's Distance* (Rubner et al., 2000), as it is currently the only measure for non-binary quantification. Results are shown on Table 4.

6. Conclusion

Sentiment analysis is very difficult task in natural language processing, but also very useful one. For this project we built simple system that classifies sentiment of a tweet on feature level. First we translated our dataset of tweets to bag-of-words vectors. Then we trained and tested our classifier using dataset that was provided by SemEval, and evaluated it using prescribed metrics. During the development of our system we occasionaly modified some of these features which improved the results.

Even though our system does not compare to the state-of-the-art systems, it passes all of the baselines. However, in this work have shown that there are some simple features and word relations that can help in determining the tweet's sentiment.

References

Andrea Esuli and Fabrizio Sebastiani. 2006. Sentiwordnet: A High-coverage Lexical Resource for Opinion Mining. In *LREC 2006, 5th Conference on Language Resources and Evaluation*, pages 417–422.

Preslav Nakov, Alan Ritter, Sara Rosenthal, Fabrizio Sebastiani, and Veselin Stoyanov. 2016. Semeval-2016 Task 4: Sentiment Analysis in Twitter. In *Proceedings of SemEval-2016*, pages 1–18.

Jeffrey Pennington, Richard Socher, and Christopher D. 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.

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, pages 99–121.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

P. D. Turney. 2002. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 417–424.

Twitter Sentiment Analysis Using Word2Vec Model

Marin Koštić, Tina Marić, Domagoj Nakić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{marin.kostic, domagoj.nakic}@fer.hr, maric.tina8@gmail.com

Abstract

This paper proposes to tackle Twitter sentiment analysis as defined by SemEval-2016 Task 4. We describe our solution to subtasks A (Message polarity classification), B (Tweet classification according to a two-point scale) and C (Tweet classification according to a five-point scale). Our approach is based on tweet preprocessing and using machine learning models like multinomial Bayes classifiers, neural networks and support vector machines to label tweets. We evaluate our model and present the results that are comparable to other task submissions.

1. Introduction

In the recent decade, social networks have become a vital part of our lives. Twitter, with more than a billion registered and more than 300 million active users, is one of the biggest virtual communities. Twitter users share short messages called tweets which are often used to convey sentiment about a certain topic. During the recent 2016 U.S. presidential election, Twitter was the go-to website for breaking news about voting with more than 40 million election-related tweets shared on the Election Day.

Sentiment analysis is the computational study of people's opinions and emotions towards given topic. It is frequently applied to examine public attitudes and is an interesting field to advertisers.

SemEval (Semantic Evaluation) is an ongoing series of evaluations of computational semantic analysis systems. Each year, a number of natural language processing problems are announced and researchers around the world tackle them. In this paper we propose a solution to Twitter sentiment analysis problem (SemEval-2016 Task 4). We have concentrated specifically on three subtasks. The first one (Subtask A) is determining whether a given tweet is of positive, negative, or neutral sentiment. For the second and the third task, besides a tweet we were also given a topic and we had to determine the sentiment expressed towards that topic, on a 2-point scale (Subtask B) and on a 5-point scale (Subtask C).

Our approach is inspired by and tries to emulate SemEval Twitter sentiment analysis task submissions and related work. Firstly, we preprocess tweets and extract potentially important features. After that we train various machine learning models such as multinomial Bayes classifiers, neural networks and support vector machines on training data. Finally, we use trained models to label tweet sentiment.

2. Related work

'What other people think has always been an important piece of information for the most of us during the decision-making process (Pang et al., 2008). Since the World Wide Web became widespread and popularization of numerous social networks such as Twitter and Facebook grew in size, the availability of sentiment analysis and opinion mining data increased. This popularized a trend of automatic detection and classification of sentiment in such freely available data. Information need has never been bigger for numerous reasons, marketers have always needed to monitor media related to their brands weather for public relations activities or competitive intelligence.

There are many forms of gathering sentiment data from user such Amazon's Star rating, and there has been a lot of research on how sentiments are expressed trough online reviews and news articles. Twitter as a target domain for sentiment analysis is full of useful sentiment information since it was designed as an community to share short informal messages with anyone. But Twitter design, although amazing for its users, for a Natural Language Processing task of Sentiment Analysis it is quite a challenging task.

In the past few years a growing number of papers have been published that tackle sentiment analysis in Twitter (Kouloumpis et al., 2011). These researchers evaluate how features extracted from text, such as part-of-speech tags, lexicon features and statistical features like n-grams, perform when applied to Twitter sentiment analysis.

Research was also done in detecting sentiment in tweets towards certain topics by topic detection methods (Cai et al., 2010). Authors use sentiment information to extract most significant topics using PMI and word support metrics, such methods not only detect sentiment, but also give an insight what is the cause and motivation behind it.

3. Implementation

In this section we describe our approach to Twitter sentiment analysis task. First step is tweet preprocessing followed by extracting potentially important features. After that we train various machine learning models on training data.

As mentioned in the introduction we have three subtasks to tackle. We decided to create two different models, one for the Subtask A, when we do not have the information about a topic, and the other one for Subtasks B and C, when we are give the topic that tweet refers to.

3.1. Dataset

Dataset was taken from the official SemEval web pages (Nakov et al., 2016) and is available for free download.

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 44–46, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

Table 1: Examples of tweets from datasets for subtasks A, B and C, respectively. Tweets are in ‘quotes. Sentiment and sentiment score are in **bold**. Target topics are in *italics*. Tweet IDs are contracted to first 5 digits

22419... **negative** ‘Dream High 2 sucks compared to the 1st one.
22686... *birthday cake* **positive** ‘Tomorrow I get to make a Birthday Cake.
64147... *netflix* **1** ‘I got my 1st achievement on the xbox one: watch Netflix 7 days in a row.

Dataset for subtask A consists of 6000 train, 1999 dev and 20632 test tweets, for each we were given tweet ID, tweet sentiment (positive, neutral or negative) and tweet text. Dataset for subtask B is similar, but additionally consists of the target topic (e.g., birthday cake), dataset consists of 4346 train and 10551 test tweets. Finally, dataset for subtask C contains score (from -2 to 2) instead of sentiment, also size of dataset is 6000 train and 20632 test tweets. Examples of provided tweets and accompanying information is given in table 1.

3.2. Preprocessing

Twitter users tend to express their thoughts using informal language. For that reason tweets differ from regular texts. They usually contain slang and misspelled words, abbreviations, emoticons and hashtags. Taking these observations into account we do the following preprocessing steps:

- text is converted to lower case,
- stop words (e.g., ‘a, ‘an, ‘and, ‘do, ‘the, ...) are removed,
- hash sign from hashtags is removed (e.g., ‘#president $\rightarrow$ ‘president),
- URLs are substituted with the string ‘URL (e.g., ‘fer.unizg.hr $\rightarrow$ ‘URL),
- user mentions are substituted with the string ‘<USER> (e.g., ‘@rihanna $\rightarrow$ ‘<USER>),
- more than three repeated characters are contracted to only three (e.g., ‘coooool $\rightarrow$ ‘coool),
- emojis are substituted with their description (e.g., :-) $\rightarrow$ ‘happy),
- every tweet is tokenized using TweetTokenizer, and resulting tokens from previous step are stemmed using SnowballStemmer (Loper and Bird, 2002).

3.3. Subtask A

SemEval-2016 Task 4, Subtask A was defined as the following:

- Given a tweet, predict whether a tweet is of positive, negative or neutral sentiment.

We approach this task by first creating corpus from SemEval tweet datasets, then apply preprocessing steps described in the section above. After preprocessing we compute two types of n-gram features based on statistical data from our corpus:

- **Word features count vectors** - Word unigrams as count based vectors (Pedregosa et al., 2011). We ignored words that have a document frequency strictly lower than two because that kind of setup gave us the best results. We also tried adding bigram and trigram features, but it did not lead to any significant improvement;
- **Word features Tf-idf vectors** - Word unigrams as tf-idf-weighted vectors (Pedregosa et al., 2011). Cut-off document frequency was also two, as well we tested with bigram and trigram features but unigrams proved as best option.

These word feature vectors were then concatenated to form a single feature vector that was later used to train machine learning models.

After feature extraction we trained the following three different machine learning models Multinomial Naive Bayes (MB), Support Vector Machine (SVM) and Multilayer Perceptron (MLP) classifier. We then used official SemEval baseline scores on subtask A and evaluated our models by doing nested 10-fold on outer loop and 3-fold on inner loop cross-validation to find best models for our three machine learning algorithms.

3.4. Subtasks B and C

Two following tasks were defined as:

- **Subtask B**: Given a tweet known to be about a given topic, predict whether it conveys a positive or a negative sentiment towards the topic.
- **Subtask C**: Given a tweet known to be about a given topic, estimate the sentiment it conveys towards the topic on a five-point scale ranging from highly negative (-2) to highly positive (2).

For subtasks B and C we further extend our feature vectors to find sentiment towards a given topic. To get additional features for tweets and topics, we decided to use word embedding vectors created using Google Word2Vec model (Mikolov et al., 2013). The model we used contains 300-dimensional vectors for 3 million words and phrases pre-trained by Google, the model was trained on Google News data set that contains about 100 billion words. We created word embeddings in the following manner:

- **Word embeddings** - To create a feature vector we represented a tweet and a topic as bags of word models. For each word that is in Word2vec model we retrieved its representation. We then summed all word representations for the tweet and also for the topic which gave us two 300-dimensional vectors. Concatenating these vectors we got final 600-dimensional word embedding vector.

Table 2: Table displays best submitted scores for subtask A, subtask baseline and our results

#	**System**	F1	Acc
1	SwissCheese	**0.633**	0.646
2	SENSEI-LIF	0.630	0.617
25	Majstor Fantac MB	0.497	0.524
35	Baseline	0.255	0.342

Table 3: Table displays best submitted scores for subtask B, subtask baseline and our results

#	**System**	ρ	Acc
1	Tweester	**0.797**	0.862
2	LYS	0.791	0.762
12	Majstor Fantac MLP	0.687	0.835
20	Baseline	0.500	0.778

This vector was used in addition with count vectors described in subtask A, we believe that such model would not only give us polarity of our tweets but word embeddings paired with statistical data from corpus would give us polarity towards a given topic.

4. Results

Finally we present our results in three tables, table 2, 3 and 4. Each table represents our best model for each of the three SemEval-2016 task 4 subtasks.

For the first subtask A we trained thee different classifiers as explained in previous section, Multinomial Bayes, SVM and Multi-layer Perceptron. After an exhaustive nested k-fold grid search over parameters the best result was Multinomial Bayes classifier whose scores are displayed in table 2. As shown our model places us at 25th place on SemEval-2016 competition. We also did a two-sided statistical t-test between our models that show that Multinomial Bayes when compared against SVM and MLP classifier is statistically significantly better at the $p < .1$ level.

Results on the second subtask B are shown in table 3. For this subtask we compared SVM and MLP classifier, and MLP classifier performed better than SVM when trained on our feature vectors that represented sentiment toward given topic on the two-point scale.

On the final subtask C, we also exhaustively trained SVM and MLP classifiers. In the table 4 we show our results. We are proud that our best model outperforms all submitted results on SemEval-2016 competition.

5. Conclusion and future work

Sentiment analysis is in general a hard problem, mostly because to perform perfectly, it requires deep understanding of text. Sentiment analysis of tweets has further specific challenges, such as poor grammar, commonly used slang, hashtags and sarcasm. Our scores and scores of SemEval-2016 Task 4 participants prove that sentiment analysis is a rather difficult assignment. Our solution is not state-of-the-art, but still places us in the middle of the ranking table. In spite of not achieving near-perfect results, we are content with our work as it was our first take on a natural language processing problem.

Table 4: Table displays best submitted scores for subtask C, subtask baseline and our results

#	**System**	MAE^M	MAE^μ
1*	Majstor Fantac MLP	**0.635**	0.633
2	TwiSE	0.719	0.632
3	ECNU	0.806	0.726
12	Baseline	1.200	0.537

For future work, we could further improve all steps of our model. Preprocessing could be advanced by analysing hashtags in more detail (some hashtags are formed from several words). Using lexicon of positive/negative words and character n-grams could be used as additional features for vector representation of tweets. Lastly, using deep learning models like long short-term memory network could boost performance of our classification.

References

Keke Cai, Scott Spangler, Ying Chen, and Li Zhang. 2010. Leveraging sentiment analysis for topic detection. *Web Intelligence and Agent Systems: An International Journal*, 8(3):291–302.

Efthymios Kouloumpis, Theresa Wilson, and Johanna D Moore. 2011. Twitter sentiment analysis: The good the bad and the omg! *Icwsm*, 11(538-541):164.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, pages 63–70. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Preslav Nakov, Alan Ritter, Sara Rosenthal, Fabrizio Sebastiani, and Veselin Stoyanov. 2016. Semeval-2016 task 4: Sentiment analysis in twitter. In *SemEval@ NAACL-HLT*, pages 1–18.

Bo Pang, Lillian Lee, et al. 2008. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135.

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.

Time Anchoring Text Using Motive Occurrence

Alen Murtić, Ante Pocedulić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{alen.murtic, ante.pocedulic}@fer.hr

Abstract

In this paper we present a solution to the Diachronic Text Evaluation task from SemEval-2015 which consists of 3 subtasks - dating a text with explicit time anchors, text with time-specific language, and evaluating whether a phrase is relevant for dating process. Using datasets provided by the competition, we tackle each one of these subtasks - offer detailed description of the problems at hand, propose our own solutions, evaluate it using official script and comment on other possible approaches. The texts provided are extracted from newspaper articles between late 1600s and 2010s. We present two solutions which use motive occurrence in the train set to date the test data, one is soft-matching motives and years, while the other uses tf-idf method. Both models performed competitively with other solutions, but neither came on top in any category.

1. Introduction

As people, their lifestyle, jobs, and environment change through time, so do ideas, politics, and language. While some would argue there are many cycles of evolution and devolution in human behavior and politics, there is no doubt language does not revert to its previous form. Because of that, as well as the fact most human knowledge is preserved in text form - both through the years and currently - older text still hold relevance even today. Classifying unknown texts and finding new data in it is something historians do regularly and it presents as an interesting problem to face.

This paper aims to solve the task of finding discernible patters in news articles through history (late 1600s - 2010s) to predict when they have been written. The task was presented at 2015 SemEval conference and a few researches have already tackled the problem. It consists of 3 separate subtasks:

1. Dating news articles which have a clear time anchor, such as - *Letters from Verona Italy say that 4000 men are hourly expected in the camp of the Prince Eugene, being most of them Troops the duke Brunswick did lately into the service of the Emperor.* - dated between 1701 and 1703.

2. Dating news articles which use time-specific language - e.g., *New corn is not yet moving freely, although some old corn that was held to see how the new crop would mature' is now coming forward.* - which was published between 1899 and 1905.

3. Detecting whether a word phrase carries information useful for dating a part of text to a specific time period. If the task is dating - *with a picket of Dragoons* - to 1740-1840 the phrase - *with a* - carries very little information, while the phrase - *picket of Dragoons* - is extremely important to the dating process.

We solved all three problems with expectedly varied degrees of success. All three subtasks have been tackled using two different methods - SVR (scikit learn.org, 2017) and our own end-to-end system. All subtasks have been evaluated using official evaluation script and compared against teams which participated at the conference competition. Section 2 will present similar problems and solutions, more on each subtask will be provided in section 3, while results will be presented in section 4. At the end, we present the final thoughts and conclusions in section 5.

2. Related work

SemEval task has referenced 4 different papers which try to tackle text-time problem:

(Wang et al., 2008) discuss detecting topics through time with their continuous time dynamic topic model (cDTM). The cDTM is built upon other works based on Latent Dirichlet Allocation, but it works as Brownian motion through document collection topics which are defined as 'a pattern of word use'.

(Popescu and Strapparava, 2013) have done an interesting job in epoch characterization using Google n-grams, a collection of n-grams which Google collected from various scientific text, news, etc. written in different years.

(Mihalcea and Nastase, 2012) is a paper whose goal most closely resembles this problem without actually focusing on it, like the next paper. It aims to disambiguate word epochs, i.e., understand how word meaning and context change over time.

(Popescu and Strapparava, 2015) directly tackles the SemEval-2015 task and all 3 of the subtasks. Concretely, it presents 4 best solutions at the conference.

- AMBRA is a 'learning-to-rank framework' which inputs pairs of texts and then extracts features - document length, style, grammar, and lexical.

- IXA is a hybrid of 4 approaches - finding the time within the text, searching for named entities, Google n-grams, and linguistic features.

- UCS uses multi-class SVM to create a stylistic classification, such as frequencies, Google n-grams, POS, etc.

- USAAR is almost a cheating system as it finds the the original texts and dates on the web - which makes it

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 47–50, University of Zagreb,
Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

score exceptionally. Even such, it does make a contribution by providing a full corpus of diachronic data.

A lot of our initial ideas have already been implemented in some manner, so we created something different. Unlike other papers which use Google n-grams, this solution is focused on extracting as much information as possible from the provided training data to solve the task, while staying in the same general idea.

3. Subtasks and solutions

For all the subtasks we have implemented 2 solutions. One using our own algorithm named Basic Timeline Classifier (BTLC) and another built on the same principle but using SVR. The idea is to lemmatize the articles, find named entities and map both with the years in which they occur. With the mapped training data, we could do the reverse search for lemmas and named entities - text elements - extracted from the unlabelled test data. Lemmas/named entities occurring often in different years have lower relevance to the final result than those which are sparse. The general idea is similar to how TF-IDF is used for web searches, ranking results using relative importance of each word in the query. We used Stanford NLTK library (nltk.org, 2017) to tokenize and remove stopwords from dataset.

3.1. Subtasks 1 and 2

As stated in the introduction, subtasks 1 and 2 have identical structure, each training input is a combination of an excerpt from a news article and a number of intervals, one of which is labelled *yes* signalizing that the excerpt is written in that interval. There are 3 versions of the intervals Fine, Medium, and Coarse, with different lengths (smallest to longest). Examining the provided data, we have concluded it has been extracted from a group of similar sources and thus our ideas would make feasible solutions.

One incredibly important note: intervals are not equal for all the inputs, i.e., one input can be Fine dated 1703-1705, while the other can have 1704-1706 as a possible interval. Thus the problem can not be framed as simple classification with predefined number of target classes.

3.1.1. BTLC classifier

BTLC is a classifier which stores data for Fine, Medium, and Coarse intervals separately. Its training phase consists of two steps - standard processing of the text, and storing the processed information:

Text processing is tokenization, PoS tagging, removal of stopwords, recognition of named entities, and lemmatization of other words. Storing means to store each text element for each year of the interval, in separate dictionaries for Fine, Medium, and Coarse intervals

After it has been trained, it can classify a text in one of its suggested intervals by firstly processing it exactly as with training, iterating through words and then returning the interval in which argmax(Prediction) falls.

For each year of the intervaledYear +/- window:

For each text element: if exists in training set:

$Prediction[year] += ocr * cnt * coeff * (window + 1 - abs(intervaledYear - year)) * 1.0/(window + 2)$

where **ocr** is the number of occurrence of the element in the year, **cnt** is the element's total number of occurrences, while **coeff** and **window** are two parameters which denote scoring value and soft boundaries (e.g., we might have texts about *Woodrow Wilson* from 1914, but it also makes him somewhat relevant to 1911) and **intervaledYear** represents a year that in in some interval (e.g., 1912, 1913, 1914 if the interval is 1912-1914)

Both values for parameters have been determined using custom implemented 5-fold cross-validation.

3.1.2. SVR method

Second approach was to use SVR method with tf–idf as a feature. Since we wanted to use SVR method, we could not use year ranges as label for data. Therefore, we used the mean of the year ranges as the label.

First we calculated tf–idf for each text in our training data with included set of stop words. After calculation, we used truncated SVD numerical method to reduce dimension of our model and remove unnecessary information. Result from truncated SVD method was our input for SVR model.

To get maximum perfomance for all our steps, we used Python Sklearn module which has implementation of *Grid Search* method for parameter optimization.(scikit learn.org, 2017)

We have experimented with only using named entities as the stored data for subtask 1, but since the named entity taggers are not perfect tools - around 90% precision (Wikipedia, 2017) - and such entities are sparse in short texts, the results were so unsatisfactory we quickly gave up on the idea.

The premise of both classifiers was that similar motives appear in similar eras and cumulatively scoring each motive according to its relevance will give us the best results. E.g. accurately dating - *Prinz Eugen marched against the Janissaries* - would be impossible without the word *Janissaries* as the rest of it could refer to the German prince (early 18th century) or Nazi army division (1940s). The same premise was applied to subtask 2.

3.2. Subtask 3

There is no special training data for subtask 3, therefore BTLC and SVR are trained in the same fashion as for subtasks 1 and 2, using both datasets at once. Data in this subtask is given as text extract + its interval + some subsets of the extract whose relevance is to be determined.

BTLC is once again used as a cumulative classifier. It first processes the subset of the text and then the whole text, after which it determines the score of both sets and scales the subset according with regards to the length of the text. Then if follows the formula:

If$(subsetScore * scaler \geq textScore)$ return yes

The subsetScore is scaled with regards to the expected textScore, as BTLC is a cumulative classifier. The idea of the method is to find out how much score does the subset carry relative to the whole text if both are of the same length. In theory, the whole text consists of important and unimportant elements. If the subset is important it should have a scaled score higher than the whole text, and the other way around.

SVR simply calculates the interval of the subtext and compares whether the subtexts fall in the predicted interval.

4. Data and experiments

4.1. Data

The data we used was provided with the task, it is in quasi-XML format for all 3 subtasks. As such, it can be processed by official evaluation script. It consists of 3 parts - training, moreTraining and goldStandard on which the model was to be evaluated. What we soon realized was that moreTraining data contained some errors - e.g., *This Day the Right Honourable John Hely Hutchinson, His Majesty's Principal Secretary of State in Ireland, was, by His Majesty's Command, sworn of His Majesty's most Honourable Privy Council, and took his Place at the Board accordingly.* - is dated between 3 B.C. and 3 A.D. when it's clear it should be dated sometime in 18th century. We removed all the bad data from the training sets.

4.2. Experimental Results

Authors of SemEval task introduced their script which calculated their defined score and precision. For each task, we should provide output file in same format as input file but with our predicted result. Result for each subtask are presented in tables 1–6. SVR has scored better than BTLC in all problems related to subtasks 1 and 2, so only its results are presented.

Experiments show that BTLC and SVR provide comparable results, as presented in tables 7 and 8. Notably, both have problems with false negatives. Our results for subtasks 1 and 2 were mostly ranked fourth our of four systems, being competitive with second and third, but only once making the jump to position three, while our result for subtask three has 6-8% lower accuracy than the two compared systems.

The only definite negative we theoretically found is the possibility of mapping evenly distributed words - such as *war*, unfortunately - to the median years of all the data.

Table 1: Subtask 1. SVR Result for $Medium$ text.

Score:	0.222
Precision:	0.0374
Distribution of predictions:	
	10 predictions which are off 0 years
	8 predictions which are off 12 years
	11 predictions which are off 24 years
	10 predictions which are off 36 years
	17 predictions which are off 48 years
	11 predictions which are off 60 years
	8 predictions which are off 72 years
	11 predictions which are off 84 years
	20 predictions which are off 96 years
	161 predictions which are off 108 years

Table 2: Subtask 1. SVR Result for $Fine$ text.

Score:	0.1083
Precision:	0.0337
Distribution of predictions:	
	9 predictions which are off 0 years
	3 predictions which are off 6 years
	5 predictions which are off 12 years
	1 predictions which are off 18 years
	10 predictions which are off 24 years
	1 predictions which are off 30 years
	6 predictions which are off 36 years
	4 predictions which are off 42 years
	2 predictions which are off 48 years
	226 predictions which are off 54 years

Table 3: Subtask 1. SVR Result for $Coarse$ text.

Score:	0.353
Precision:	0.00636
Distribution of predictions:	
	17 predictions which are off 0 years
	18 predictions which are off 20 years
	25 predictions which are off 40 years
	15 predictions which are off 60 years
	31 predictions which are off 80 years
	0 predictions which are off 100 years
	9 predictions which are off 120 years
	20 predictions which are off 140 years
	3 predictions which are off 160 years
	129 predictions which are off 180 years

Table 4: Subtask 2. SVR Result for $Medium$ text.

Score:	0.5275
Precision:	0.0492
Distribution of predictions:	
	51 predictions which are off 0 years
	53 predictions which are off 12 years
	109 predictions which are off 24 years
	43 predictions which are off 36 years
	203 predictions which are off 48 years
	231 predictions which are off 60 years
	170 predictions which are off 72 years
	59 predictions which are off 84 years
	56 predictions which are off 96 years
	66 predictions which are off 108 years

Table 5: Subtask 2. SVR Result for $Fine$ text.

Score:	0.208
Precision:	0.0347
Distribution of predictions:	
	36 predictions which are off 0 years
	22 predictions which are off 6 years
	48 predictions which are off 12 years
	13 predictions which are off 18 years
	84 predictions which are off 24 years
	13 predictions which are off 30 years
	21 predictions which are off 36 years
	142 predictions which are off 42 years
	102 predictions which are off 48 years
	560 predictions which are off 54 years

Table 6: Subtask 2. SVR Result for $Coarse$ text.

Score:	0.7469
Precision:	0.0559
Distribution of predictions:	
	58 predictions which are off 0 years
	130 predictions which are off 20 years
	184 predictions which are off 40 years
	317 predictions which are off 60 years
	218 predictions which are off 80 years
	100 predictions which are off 100 years
	29 predictions which are off 120 years
	0 predictions which are off 140 years
	0 predictions which are off 160 years
	5 predictions which are off 180 years

Table 7: Subtask 3. Confusion matrix for SVR.

n=383	Predicted: yes	Predicted: no
Actual: yes	68	108
Actual: no	84	123

Table 8: Subtask 3. Confusion matrix for BTLC.

n=383	Predicted: yes	Predicted: no
Actual: yes	59	116
Actual: no	83	125

Many false negatives that occur in subtask 3 are likely due to the small corpus with which we have worked. In theory, using Google n-grams should fix the issue.

5. Conclusion

In this paper we have presented a new approach to solving SemEval 2015 task 7 as well as the general diachronic text evaluation problem. Mapping the words and their occurrence is something most teams considered, but using different algorithms. We proposed mapping the center of each interval as an output to an SVR which takes tf-idf weighted vectors of each text element - lemma or named entity - as the input. SVR was tested against systems in the competition as well as our heuristic baseline.

We then proposed using the same SVR as the classifier for determining whether a word carries relevant information to the dating problem and tested it on the data provided by the task. Experimental results show that SVR outperforms our own classifier.

We note that our solution works fairly well when it comes to provided data and it might work better on a larger scale, using Google n-grams and trying to date a large corpora of text, which will provide more accurate word counts. Nevertheless, even on this scale, it has been shown that this approach does offer promise and therefore could warrant further exploration of the method.

Acknowledgements

We express gratitude to TakeLab staff for helping us map intervals with numbers and suggesting tf-idf approach.

References

Rada Mihalcea and Vivi Nastase. 2012. Word epoch disambiguation: Finding how words change over time. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 259–263, July.

nltk.org. 2017. Natural language toolkit.

Octavian Popescu and Carlo Strapparava. 2013. Behind the times: Detecting epoch changes using large corpora. *International Join Conference on Natural Language Processing*, pages 347–355, October.

Octavian Popescu and Carlo Strapparava. 2015. Semeval 2015, task 7: Diachronic text evaluation. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 870–878, June.

scikit learn.org. 2017. scikit-learn: machine learning in python — scikit-learn 0.18.2 documentation.

Chong Wang, David Blei, and David Heckerman. 2008. Continuous time dynamic topic models. *In Proceedings of the International Conference on Machine Learning*, July.

Wikipedia. 2017. Named-entity recognition.

An Ensemble-Based Approach to Author Profiling in English Tweets

Mihael Nikić, Martin Gluhak, Filip Džidić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{mnikic777,mr.gluhak,filipdzidic}@gmail.com

Abstract

Author profiling is the task of extracting personal information about an unknown author of a given text. This paper describes the use of a supervised learning model for classifying authors of English tweets based on their age and gender. The dataset is provided as a part of the PAN 2016 challenge. The experiments are performed with SVM as a baseline model along with other common ML models used for classification using various features which are extracted from tweets in the dataset. The models are evaluated using accuracy and F1 score.

1. Introduction

Author profiling is the task of determining information about an unknown author of the given text. The main categories of the author which are often extracted are age, gender, and personality. This is usually done by using various NLP and machine learning techniques and models. This is becoming increasingly important and useful in various fields such as forensics, marketing, and security.

We only focus on finding the age and gender of the author based on their tweets as defined in the PAN 2016 event (Rangel et al., 2016). As a baseline model to this problem we use a bag-of-words (BOW) model that represents text as sparse numerical feature vectors. To downweight frequently occurring words in the feature vectors, we use a term frequency-inverse document frequency (tf-idf) technique. The models were evaluated using standard performance metrics, more specifically we measured the accuracy and F_1 score.

This paper is organized as follows: Section 2 mentions other related work done on this task as well as how it influenced our approach to this problem, Section 3 describes the dataset used for testing and training as well as the preprocessing methods used on it, Sections 4 and 5 describe the feature engineering and the ML models we use. Section 6 describes the evaluation and testing in detail and we finish off with a conclusion in Section 7.

2. Related Work

Author profiling has been a staple task at the PAN contest for several years. In this time there have been several outlined approaches to solving this task. Some of the competitors make use of ensemble learning in their models as this has shown to give better final performance compared to using a single trained classifier (Agrawal and Gonçalves, 2016).

Last year's winners op Vollenbroek et al. (2016) experimented with using second order attribute feature, but concluded they do not bring much improvement on author profiling of the the twitter domain. The intention was to build a model that would generalize well in order to perform on different domain documents which were a part of the PAN test dataset. A portion of the features we used were inspired by this paper, although some of them were altered.

Table 1: PAN 2016 corpus.

Trait	Label	Number of authors
Age	18–24	28
	25–34	140
	35–49	182
	50–64	80
	65+	6
Gender	Male	218
	Female	218
Total		436

Some contestants have tried to analyze the linguistic style of authors by using stylometric features (Pervaz et al., 2015; Ashraf et al., 2016). The idea is to identify author writing traits with the expectation that his style will not change across different domain documents.

3. Dataset and Preprocessing

We used the PAN 2016 dataset which consists of tweets from 436 different authors. They are labeled by gender and age groups. The age groups are 18–24, 25–34, 35–49, 50–64, and 65+. Detailed distribution is shown in Table 1. The genders are equally distributed while some of the age groups are underrepresented. This dataset was originally called the training dataset because in the PAN competition there where several different test dataset with documents from different domains. We were unable to retrieve these test documents so we used the available dataset for evaluation.

The used dataset consists of several documents. Each document represents a single author and contains all of his written tweets. Every document was converted into a key-value pair where value is represented as a list of user tweets and user ID was used as the key which identifies each user. After tokenization we replaced unneeded tokens with placeholders (URLs and numbers) and additionally, we removed all HTML tags.

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 51–54, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

4. Features

4.1. Dense Word Representation

Word embeddings are words transformed into decimal numbered vectors of smaller dimensions (dense vectors). Combining several words into one vector can be done by different methods (e.g. adding vectors or finding the element-wise mean/maximum/minimum), but the performance drops when the number of combined words increases. Since each author in the dataset has many tweets it is to be expected that one word embedding that describes the author may not produce very good results as too many dense vectors are combined into only one. To experiment with this we have extracted features from vectors of a pretrained Stanford Twitter GloVe model[1] (GloVe is an algorithm for obtaining word embeddings). Interestingly, training the model using only these features performs on par with the baseline.

We have also created our own word2vec model from the PAN dataset to see how it performs. As expected, it performs worse than the pretrained model, which is reasonable as it was trained on a dataset orders of magnitude bigger.

4.2. Sparse Word Representation

We have used the bag-of-words model in combination with term frequency-inverse document frequency (tf-idf) to produce sparse word features. Usually, word embedding models are better at capturing semantic similarity than sparse vectors, but considering the previously stated problems, sparse models might be the preferred method.

4.3. Other Features

Sentence Length

We measure the average length of a sentence expressed in the number of words it contains with the assumption that older authors tend to use longer and more complex sentences compared to younger authors.

Word Length

We measure the average word-length over all the words used by a given author expressed by the number of characters they contain. The assumption is that older authors will use more complex words compared to younger authors.

Capitalized Letters

Our dataset, sourced from social media, does not contain grammatically perfect text. We assume that older authors tend to be more careful about proper capitalization which is expressed by this feature. We measure what proportion of the author's sentences start with capital letters.

Capitalized Tokens

Similarly to the previously mentioned feature, we assume that younger authors will also be less concerned with proper capitalization of tokens in their sentences. In this feature we measure the proportion of capitalized tokens in a sentence, averaging the result over all sentences for a given author.

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

Emoticons

We define emoticons as any multicharacter tokens which start with ':','=' or ';'. Authors of different ages tend to use emoticons in different ways. Younger authors prefer noseless emoticons such as ':)' compared to older authors who tend to give their emoticons nose characters like in ':-)'. Younger authors are also overall more likely to use emoticons compared to older authors. We measure the frequency of used emoticons as well as how many of them have noses.

Ends with Punctuation

We measure the proportion of sentences which end with punctuation as younger authors tend to not punctuate all of their sentences in social media.

Out of Dictionary Words

We measure the number of misspelled words out of total number of words per author with the goal of distinguishing different age groups. By misspelled words we mean all words that could not be found in a typical English dictionary, this includes slang, brand names and Spanish words.

N-grams

We experimented with different n-gram sizes for the creation of sparse tf-idf matrices combined with feature selection.

Punctuation by Sentence

We measure the overall use of punctuation in sentences by counting the occurrences of the most common punctuation characters (",", ".", ";", "!", "?" and "-") in each sentence as a proportion of all the characters in the sentence. This is then averaged out over all the sentences of a given author.

Vocabulary Richness

We assume that older authors are more likely to have a broader vocabulary compared to younger authors. We measure vocabulary richness by counting how many words used by an author were only used once. The total number of these unique words is then divided by the total number of different words.

POS Tags

We extracted the part-of-speech tags and calculated the frequency of each tag appearing in the tweets for individual authors.

5. Models

5.1. Baseline

The baseline utilizes the SVM classifier as it is successful and widely used in the natural language processing problems. It only uses the features derived from the bag-of-words model combined with the tf-idf.

5.2. Additional Features

This model also uses SVM as the classifier and combines features from the baseline model with additional features specified in 4.3.

Table 2: Performance results of the models used in the paper. Macro-averaged scores are shown.

Model	Trait	Accuracy	Precision	Recall	F1
Baseline	Age	0.486229	0.242697	0.262694	0.235315
	Gender	0.718340	0.718447	0.718340	0.718303
	Both	0.326105	0.231609	0.207512	0.199005
BOW + Features	Age	0.472685	0.273523	0.276205	0.257404
	Gender	0.720507	0.721251	0.720507	0.720144
	Both	0.325944	0.269363	0.216314	**0.214065**
Stacking	Age	0.488764	0.270331	0.261552	0.238127
	Gender	0.768605	0.769578	0.768605	**0.768383**
	Both	0.208917	0.113021	0.143210	0.126337
GloVe	Age	0.423206	0.291020	0.275993	**0.263360**
	Gender	0.767314	0.777206	0.766879	0.765233
	Both	0.272767	0.200626	0.182756	0.178487

Table 3: Base and meta classifiers used for classification.

Category	Base classifier	Meta Classifier
Gender	Bayesian LR Multinomial NB Naive Bayes Linear SVM	Naive Bayes
Age	Multinomial NB Logistic Regression Naive Bayes Linear SVM	Linear SVM
Both	Gender Classifier Age Classifier	Voting Classifier

5.3. GloVe

The word embeddings are extracted from the 100-dimensional Stanford GloVe pretrained model. Features are created by calculating the average of embedding vectors from all the words used by the same author. These features are combined with the feature set from 5.2.

5.4. Stacking

Stacking is an ensemble learning method which combines outputs of multiple classification models as input to the classifier that gives a final output – meta-classifier. The individual classification models are trained on the PAN dataset and then, the meta-classifier is fitted based on the outputs of the individual classification models in the ensemble.

In our approach we decided to use an existing python library for stacking called *StackingClassifier*.[2] Table 3 shows classifiers used in our approach. The base and meta classifiers were chosen based on the work by Agrawal and Gonçalves (2016)

6. Evaluation

The models were evaluated using a nested 5-fold cross validation. Inside the cross validation the hyperparameters were optimized by using grid search. Only the *C* hyperparameter of the scikit-learn linear SVM was optimized.

The baseline performed fairly well in comparison to other models we have implemented. Stacking performed the best on the gender trait, GloVe performed best on the age trait, while the BOW + Features model performed best on both traits simultaneously. Interestingly, every model produced the best results for one trait only, but we believe that it was purely a coincidence and the bias of the dataset.

The results do not seem to be very high, but most of our work was built upon papers which tried to implement models that generalize well in order to get good results on the different domain datasets. It is hard to tell whether our models would translate well onto different datasets from the PAN competition, as they are not available.

7. Conclusion

The GloVe approach did not work particulary well because the way we represented all the tweets of individual authors in one embedding loses a lot of information. Stacking also did not yield very significant improvements, but it does heavily depend on the quality of the used classifiers.

In conclusion, author profiling has proved to be a challenging task. We experimented with stacking, word embedding and various additional features and in the end the results did not show great improvements compared to the baseline model. Previous publications done on this task show that a bag-of-words model combined with SVM works well as the baseline but there is still room for improvement.

References

Madhulika Agrawal and Teresa Gonçalves. 2016. Age and gender identification using stacking for classification notebook for pan at clef 2016.

Shaina Ashraf, Hafiz Rizwan Iqbal, and Rao Muhammad Adeel Nawab. 2016. Cross-genre author profile prediction using stylometry-based approach. *Balog et al.[5]*.

Mart Busger op Vollenbroek, Talvany Carlotto, Tim Kreutz, Maria Medvedeva, Chris Pool, Johannes Bjerva, Hessel

[2] https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

Haagsma, and Malvina Nissim. 2016. Gronup: Groningen user profiling.

Ifrah Pervaz, Iqra Ameer, Abdul Sittar, and Rao Muhammad Adeel Nawab. 2015. Identification of author personality traits using stylistic features. In *Proceedings of CLEF*.

Francisco Rangel, Paolo Rosso, Ben Verhoeven, Walter Daelemans, Martin Potthast, and Benno Stein. 2016. Overview of the 4th author profiling task at pan 2016: cross-genre evaluations. *Working Notes Papers of the CLEF*.

Dependency-Based Approach to Semantic Text Similarity

Domagoj Polančec, Branimir Akmadža, Damir Kovačević

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000, Zagreb, Croatia
{domagoj.polancec,branimir.akmadza,damir.kovacevic}@fer.hr

Abstract

This paper describes an approach to determining the semantic equivalence of two sentences which uses dependency parsing. The model presented in this paper is quite simple and only relies on twelve features obtained from semantic compositions of skip-gram vectors. Even though the model is simple it attains acceptable results when tested on English sentences which were also used to test the submitted solutions at the 2017 SemEval competition. The skip-gram vectors used are generated by pre trained word-to-vec model made available by Google.

1. Introduction

Semantic text similarity is a degree of equivalence between two pairs of text. (SemEval, 2017) While determining this measure may sound like a simple problem to solve it is actually quite difficult since humans can percieve sentences that are almost identical as considerably different. For example, let us consider this pair of sentences: *"The girl has a driver's license"* and *"A man has a driver's license."*. The difference between these two sentences is in only two words: an article and a noun with all other information pertaining to the subject unchanged yet a human annotator provided a similarity score of only 2.6 out of 5 (SemEval, 2017) which would imply that there is only slight similarity between these two sentences. It turns out that depending on where in the sentence the change is made and what was the function of the changed word, the transformed sentence can look almost the same, but bear the opposite meaning. One of the ways of determining the exact function of the word in a sentence and how it relates to other words is dependency parsing.

In this paper we devised a feature-extraction strategy which heavily relies on dependency parsing. As an NLP framework we used NLTK library (Bird et al., 2009) together with Stanford Dependency Parser (de Marneffe et al., 2006). The models we tested in this paper are a Support Vector Regression model and a Multi-Layer Perceptron model. We used scikit-learn (Pedregosa et al., 2011) implementation of Support Vector Regression model, as well as its grid search optimizer. Multi-Layer Perceptron model was created in Keras (Chollet and others, 2015) with Tensorflow backend (Abadi et al., 2015). As an initial baseline we used cosine similarity computed over sum of the skip-gram vectors obtained by simply tokenizing the sentences and feeding the tokens to the word-to-vec model. Later we also applied preprocessing to the sentences before tokenizing them and computed a new baseline the same way.

2. Related Work

A lot of previous work was done on semantic text similarity tasks and many approaches have been tested. We analyzed two very different approaches to the task. The sentence based similarity score can be computed using a support vector regression (SVR), combining a lot of different features was proposed as a solution for SemEval 2012 (Sarić et al., 2012). Another approach using probabilistic soft logic to combine logical and distributional representations of natural-language meaning was also explored (Beltagy et al., 2014).

For the SVR approach, the datasets were preprocessed using an 8 step method combining standard preprocessing steps with several newly proposed. Several key features were extracted using knowledge-based and corpus-based similarity. Cross-validation identified several key features that improve the score on any dataset and some other that are domain dependent. The system performed very well in general, especially on shorter sentence datasets.(Sarić et al., 2012)

The second system improved on a previous system proposed by the same authors. They use logic to capture sentence structure, but combine it with distributional similarity ratings at the word and phrase level. They improved upon the previous system by switching Markov logic networks for the probabilistic soft logic (PSL). PSL defines a distance to satisfaction for the expression and the optimization problem becomes a linear problem, thus boosting computational performance significantly. Evaluation on three datasets shows that it also improves on the basic semantic text similarity approaches and the previous approach using Markov logic networks.(Beltagy et al., 2014)

An approach to semantic compositionality proposed by Socher et al. (2013) was also explored. This approach treats adjectives and adverbs as linear operators upon nouns and verbs that they modify and proposes a way of computing a matrix that best corresponds to that linear operator within context of the given corpus (Socher et al., 2013).

In our work we mostly rely on ideas proposed by Sarić et al. (2012) and try to simplify the approach proposed by Socher et al. (2013). We did not attempt to incorporate PSL into our approach.

3. Preprocessing

In our research we found that preprocessing can have a significant impact on the performance of our models. The Pearson's correlation coefficient of the baseline model in-

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 55–59, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

creased from 0.72 to 0.76 after preprocessing was applied. Our preprocessing pipeline consists of the following steps:

1. **Turn all characters to lowercase** - this step is performed to simplify the input text and extract away noisy information such as beginnings of news headlines being written in all-caps. A downside of this step is that performing Named Entity Recognition is much harder and indeed, we chose to ultimately not use Named Entities as features.

2. **Apply first round of regex replacements** - this step replaces all character sequences that match a certain regular expression with the predefined string. For example, words that match regular expressions such as "don.t", "won.t", "can.t", where "." denotes a single occurence of any character, are replaced by words "do not", "will not", "can not" and so on. This is done because in the training dataset, it was often the case that the common apostrophe (') was replaced with either a slanted apostrophe (‘) or a question mark (?). Similar step was also performed by Sarić et al. (2012).

3. **Apply second round of regex replacements** - this step replaces some non-letter characters such as hash tags (#), parentheses, tildes (˜), interpunction marks, mathematical operators (+, *) and hyphens with either a blank space (parentheses, apostrophes, hyphens and hashtags) or nothing (interpunction marks). Whether a character was replaced by a blank space or nothing was made after closer inspection of the training dataset and in the interest of maximizing information obtained from the dataset (e.g., we noticed that hashtags were concatenated together in some sentences and we wanted to use words marked with hashtags when building features).

4. Feature Extraction

The feature extraction consists of the following steps: the Natural Language Processing (NLP) step, the composition step, the reduction step, and the post-processing step. These steps are described in detail in the following subsections.

4.1. Natural Language Processing Step

The first step in the feature extraction is to apply Natural Language Processing tools to the preprocessed text. This includes dependency parsing and lemmatizing. First, each sentence in the pair is parsed using Stanford's Dependency Parser. This yields the following information for each token in the sentence: the words position in the sentence, the word's dependencies in the sentence (e.g. which word is the object for a root or which word determines the subject), the word's function in the sentence (a root, a subject or something else), and a Part-of-Speech (POS) tag for that word. The raw word and its POS-tag are then fed to the WordNet based lemmatizer available in the NLTK library which determines the word's lemma. This process is repeated for each sentence in the pair of sentences and for each pair in the dataset.

4.2. Composition Step

In this step, the information from the NLP step is used to obtain a feature vector which describes the sentence. This vector should capture the semantic meaning of the sentence and the process of obtaining that information from individual words in a sentence is called semantic composition (Turney, 2014). First, we represent every word in the sentence except for stop words as a vector. We observed an increase in performance when we skipped stop words versus when we did not skip them. We use the standard English language list of 175 stop words provided by Ranks.nl (Ranks.nl, 2017). We are using Google's pre trained word-to-vec model to obtain vectors for each word. Our approach to semantic composition is a simplification of the approach outlined by Socher (Socher et al., 2013). Socher treats adjectives, adverbs and other modifiers as linear operators that act upon the nouns and verbs that they modify (Socher et al., 2013). This means that modifiers should be represented as matrices. Socher uses a more complicated way of computing such matrices, but in our case the matrix corresponding to the given modifieris given by:

$$M = b^T \cdot b \tag{1}$$

In equation (1) the b is the row word-to-vec vector of the modifier and M is the modifier matrix. The vector of the word that is being modified is left-multiplied by all matrices of the modifiers that modify that word. After modified vectors are computed, all object vectors are added together to form one vector which represents all the objects in the sentence, the same is done for subject vectors (if more are present) the root vectors (only conceptually since there can only be one root vector) and the vectors which are neither the object nor the subject nor the root are added together to form a vector that represents all the words which are not subjects, objects or root words. Finally, another vector is computed: the weighted sum of all the words in the sentence where the weights are determined by the word's POS tag, namely, the weight for the given word will be different if it is tagged as a noun, a verb, an adjective or an adverb (stop words are skipped). This gives an alternative semantic composition. These weights were optimized with a genetic algorithm.

4.3. Reduction Step

The 300 dimensional skip-gram vectors generated by the Google News word-to-vec model were shown to be too large to be used as features themselves in our models. Therefore, in the reduction step, we extract 12 features from the skip-gram vectors obtained in the previous two steps. The first two features are the distance between the weighted sentence vectors of the two sentences in the pair and the cosine similarity of the weighted sentence vectors. The next eight vectors are the same two numerical values extracted from the four vectors obtained in composition step. The last two features are the measures of overlap between sentences with regards to different types of tokens. The first measure is the overlap of numerical tokens (numbers) and the second measure is the overlap of non-numerical tokens (words) in the sentences. These measures are calculated as

Table 1: Considered corpora and training set words coverage.

Corpus	Percent of training set words coverage	Examples of words that are not covered
webtext(nltk)	19.0	finance, crumble, outward
brown(nltk)	42.5	refine, montenegro, kosovo
reuters(nltk)	31.9	elusive, horror, dose
google-news	79.2	armour, impregnations, developper

proposed by Sarić et al. (2012) and shown in equation (2).

$$overlap = \frac{2|A \cap B| + \epsilon}{|A| + |B| + \epsilon} \quad (2)$$

In equation (2) *A* and *B* are sets of numbers or words in the first and second sentence in the pair respectively and ϵ is some small number to prevent division by zero when sets are empty (e.g. when there are no numbers in either of the sentences). After reduced features have been calculated for every word in the dataset, the mean of each feature is subtracted and the features are scaled to unit variance.

5. Corpus Selection for Word Embeddings

Since the distance between two words was calculated using word vectors, we needed to generate them. For that purpose our first choice was to use package *nltk.corpus* and its corpora. We also considered a pre-trained word-to-vec model which was trained on part of the Google News dataset (Google, 2013). To find out which corpus was right for our problem, we checked how many words from training set were present in each of the considered corpora. Table 1 shows how many of unique words from training set was covered by each considered corpus. All words from training set were lemmatized.

Total number of words in training set was 147817 out of which there were 10471 unique words. Table shows that the Google News corpus used to train word-to-vec model provided by Google had best coverage of training set words which was expected because it is the largest corpus, and other corpora were made for specific purposes.

6. Models and Training

Both models are trained on the same training dataset and features extracted as outlined in Section 4. The training dataset consists of some of the training and test datasets from the previous SemEval competitions, namely the training datasets from the 2012 SemEval competition and the test datasets from the 2014 SemEval competition (SemEval, 2016).

These datasets contain various styles of writing ranging from news headlines over dictionary entries to forum posts. In total there were 5984 sentence pairs and annotations in the dataset. Each pair of sentences is given a human determined numerical similarity score where 0 is the lowest score, indicating no similarity and 5 is the highest score indicating complete similarity (SemEval, 2017).

The SVR model uses RBF kernel and its C and γ hyperparameters were optimized using exhaustive gridsearch on a random smaller subset of the training dataset with nested 3-fold cross-validation. The margin penalty hyperparameter epsilon was not optimized and was set to 0.1. The optimal pramaters found by the optimization are: $C = 2^4$, and $\gamma = 2^{-6}$.

Table 2: Parameters of the genetic algorithm used to optimize weights in weighted sum of vectors.

Parameter	Value
type	steady-state
population size	6
selection	3-tournament
mutation chance	0.99
mutation	gaussian
mutation sigma	0.01
crossover	SBX

The optimal weights for the weighted sum of vectors were found using a genetic algorithm with parameters set up as shown in Table 2. The evaluation function for the algorithm used the weights to extract features from sentence pairs belonging to a smaller subset of the training dataset and trained an SVR model on those features and sentence annotations. The fitness value is the Pearson's correlation coefficient between the output of the SVR and the annotations for that part of the training dataset.

The Multi-Layer Perceptron model's architecture is as follows: an input layer of 12 nodes, followed by a dense layer of 128 nodes which is followed by a dense layer of 64 nodes which is followed by a single node in the output layer. The activation function for neurons in hidden layer is $tanh$, the neurons in input and output layer have linear activations. This architecture was not optimized automatically but was obtained through trial and error and from our own experience. The loss function is the mean squared error. The model is trained over 4000 epochs.

7. Evaluation

Models were evaluated on the same dataset that was used to evaluate submitted entries to the SemEval 2017 competition on the English-English track. This dataset contains 250 pairs of sentences together with annotations. The competition organizers use Pearson's correlation coefficient as fitness metric. The organizer's baseline is set at 0.72 and the best entry on the English-English track had 0.85 Pearson's correlation coefficient (SemEval, 2017). In addi-

Table 3: Results of model evaluations. Please note that baseline models were *not* evaluated on the reduced features used for SVR and MLP nor the weighted sum of vectors. They were evaluated on the basic sum of all vectors in the sentence.

	SemEval baseline	our baseline	our baseline (prep.)	SVR	MLP	SemEval2017 winners
Pearson's cor.	0.72	0.72	0.76	0.77	0.72	**0.85**
Spearman's cor.	-	-	**0.79**	**0.79**	0.73	-
MSE	-	6.54	4.84	**1.18**	1.22	-

tion to Pearson's coefficient, we tested our models against two additional metrics: Spearman's correlation coefficient which, unlike Person's coefficient, does not assume linear correlation between variables, and Mean Squared Error. The results of evaluation of our models are shown in Table 3.

It seems that SVR outperforms MLP on all metrics, however, MLP has much more hyperparameters that need to be tuned and, given more time, it might have been possible for the MLP to outperform SVR.

The baseline models are quite close to the best performing models when it comes to Pearson's and Spearman's correlations, however, they lag behind when Mean Squared Error is taken into account. When inspecting the scores assigned by the models the general impression was that the scores given by the models were more realistic and more in line with what we would score certain pairs than the scores of the baseline models. The scores of the baseline models rarely dropped below 3.0 while the scores of the models were more spread out.

8. Future Work

It would be interesting to further explore the possibilities of using features from dependency parsing in models tackling the sementic text similarity scoring as those features seem to have performed well even with small number of features and on simple models.

Since our models are so close in terms of Mean Squared Error, further work is needed to differentiate them. Specifically, more time should be spent tuning the hyperparameters of the MLP model as well as statistical analysis should be performed when models are trained on other, more diverse, datasets.

9. Conclusion

In this paper we tested two simple models on the task of assigning semantic text similarity scores to pairs of sentences. The models we tested: SVR and MLP performed reasonably well on the dataset comprised of old training and test data. These models really seem to have benefited from the addition of features extracted from dependency parsing. Even though the models performed well, they still have not performed significantly better than the most basic cosine similarity which was used as a baseline in terms of Pearson's coefficient. However, they significantly otperformed the baseline in terms of Mean Squared Error.

The SVR model performed better than the MLP model in all metrics, however, in tearms of Mean Squared Error, the two models are quite close and further testing is needed to differentiate their performance.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Islam Beltagy, Katrin Erk, and Raymond Mooney. 2014. Probabilistic soft logic for semantic textual similarity. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

Francois Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *LREC 2006*.

Google. 2013. Google code - archive - word2vec. https://code.google.com/archive/p/word2vec/.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Ranks.nl. 2017. Stopwords. http://ranks.nl/stopwords.

Frane Sarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. 2012. Takelab: Systems for measuring semantic text similarity. In *First Joint Conference on Lexical and Computational Semantics (*SEM)*.

SemEval. 2016. Semantic textual similarity ¡ semeval-2016 task 1 - data and tools. http://alt.qcri.org/semeval2016/task1/index.php?id=data-and-tools/.

SemEval. 2017. Semantic textual similarity ¡ semeval-2017 task 1. http://alt.qcri.org/semeval2017/task1/.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christo-

pher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Peter D. Turney. 2014. Semantic composition and decomposition: From recognition to generation. *National Research Council Canada – Technical Report*.

Sexual Predator Identification and Suspicious Lines Extraction Using Support Vector Machines

Dario Sitnik, Matej Crnac

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{dario.sitnik,matej.crnac}@fer.hr

Abstract

This paper describes our approach to the sexual predator identification problem which was presented at PAN 2012 competition. The given problem has two tasks. The first is to identify predators among all users in given conversations and the second is to identify the parts of predator conversations which are most distinctive of predators' bad behavior. We present a solution that uses two classifiers. One is used for detecting suspicious conversations and the other for identifying predators. For classifiers, we use SVM with TF-IDF vector space and some additional features. We compare our results with those presented at PAN 2012 competition. For the first task, we would be placed third and for the second task, we would have the best score.

1. Introduction

Social networks like Facebook, Instagram or Twitter have become a major part of everyday life. The latest studies show that people spend more than two hours a day on social networks. This effects society in many ways, some positive and some negative. One of the most dangerous effects is easier communication between sexual predators and their victims. This has become a significant problem and there is a need for systems that can identify predators and make it easier for the police to catch them.

The paper describes our approach to implementing a system for identifying predators. This problem is a part of PAN 2012 competition. It consists of two tasks. The first is to identify sexual predators in conversations and the second is to identify the lines of predators' conversations which are most distinctive of predators' bad behavior. For the first problem, the participants were given a labeled training set which consists of 66,928 conversations with 97,690 users of which 148 are labeled as predators. The second task does not contain labeled data, so solving the first problem is necessary before starting with the second one.

Other works which tackle this problem are presented in Section 2. In Section 3., we present our methods for solving both tasks, and in Section 4. we evaluate our results and compare them with the results presented in PAN 2012 competition. Finally, in Section 5. we present the conclusion of this paper.

2. Related work

As this problem was a part of PAN 2012 competition, we can reference to some of the approaches presented therein. As mentioned in (Inches and Crestani, 2012) the best result for the first task was achieved by Villatoro-Tello et al. (2012), whose $F_{0.5}$ score was 0.935. They divided the first task into two main stages: the Suspicious Conversations Identification stage and the Victim From Predator disclosure stage. The first stage was used as a filter which identifies suspicious conversations. The second stage was used to identify the predator from the suspicious conversations. A major part of their successful solution was a prefiltering stage in which they removed conversations that they categorized as unnecessary.

Another interesting approach was presented in (Parapar et al., 2012), where they used psycholinguistic features. Psychology shows that words people use in their daily lives can suggest persons' social behavior. Based on that, they calculated a degree to which people use different categories of words.

Regarding the second task, the best solution was presented in (Popescu and Grozea, 2012), with F_3 score of 0.476. The number of returned lines was 63,290, of which 5,790 were correct. Because the official score for evaluating the result was F_3 score, which favors recall, they were placed first. Apart from this straightforward solution, the method used most was filtering the lines of predators using some kind of score. The third solution was to make a language model of parts of predicted predators' conversations.

3. Proposed method

Given all proposed solutions from the related work, we tried to accomplish the best results via prefiltering conversations, using various feature extractors, comparing classifiers using cross-validation and optimizing hyperparameters using nested k-fold cross validation with 5 folds. We present the solutions for both tasks which are described in the following subsections. The system is implemented using Python and scikit-learn package (Pedregosa et al., 2011).

3.1. Sexual predator identification

Just like it was proposed in (Villatoro-Tello et al., 2012), we also decided to divide the first problem into two tasks. The first is to extract suspicious conversations. The conversation in training set is considered suspicious if it has a labeled predator participating in the it. The second task is to differentiate between victim and predator in every suspicious conversation. Obviously, the solution to the second problem depends on the state of the suspicious conversations detected.

Text Analysis and Retrieval 2017: Course Project Reports (TAR 2017), pages 60–63, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, July 2017.

Table 1: Dataset size before and after preprocessing.

	Original data	Preprocessed data
Train set conversations	66927	8045
Test set conversations	154068	18600

3.1.1. Preprocessing

The given corpus contains many messages that are not useful while training classifiers. Furthermore, those messages could lead the given classifier to wrongly identify whether a user is a sexual predator or not. Assuming the predator wants the victim to stay involved in the conversation, it is less likely that the predator will post URLs to distract the victim. Also, there are many spam messages containing URLs, ASCII-Art or sequences of punctuation characters. Therefore, conversations containing mentioned messages should be excluded. Also, if the conversation does not contain exactly two users - it is not relevant. It is not likely that a sexual predator would approach a user if there are more people in the conversation. Characters of reduced training corpus are then once again analyzed and if applicable, replaced. For instance, HTML elements such as *&apos* have been replaced by apostrophe.

Furthermore, it is necessary to create the training dataset for the suspicious conversation detection. Every line within the conversation is joined and marked positive if it contains predator users. The training dataset for the second task is established by joining together all user-specific lines in suspicious conversations and marking them as positive if a user is predator and negative if not.

3.1.2. Feature extraction and classification

As mentioned in Subsection 3.1.1., the first part of the problem is to carry out the binary classification and to divide the corpus into suspicious and non-suspicious conversations. For this, we tried a combination of various classifiers and different feature extractors. SVM classifier with linear kernel, K-Nearest Neighbors with 7 and 43 neighbors, Multi-layer Perceptron classifier and Random Forest classifier were combined with Bag of Words and TF-IDF weighting .

Every classifier-vectorizer combination was evaluated and tested on the validation set. After the best combination was chosen, the pipeline containing vectorizer and classifier was optimized using Nested 5-fold cross validation. The pipeline with its optimized hyperparameters is then chosen to do the suspicious conversations classification. Similarly, the same method is used on the second part of the problem (victim vs. predator) with two additions. Firstly, features are enriched with a number of emoticons, a number of question marks, and a number of conversations started by the user. Unlike the first part of the problem, the standard scaler does not affect the classifier's performance negatively, so we included it in our pipeline.

3.2. Distinctive predator lines identification

As mentioned in Section 2., there are three main approaches to identify the lines which reveal sexual predators. The first is to simply pass all lines to the evaluator, the second is to build a language model, and the third is to do the TF-IDF weighting. Given the F_3 score as a metric, it is wise to give as many lines as possible because it is likely to get the recall high. However, besides passing all lines to the evaluator we tried to pass N-best lines based on the TF-IDF weighting of items in n-gram sequence of the line. N-gram sequence is made of line unigrams up to four-grams as shown in Table 3. Using all detected predators, we extracted their lines from the dataset. Then, the TF-IDF weighting was applied and items in n-gram sequence were sorted by ascending order according to the number of occurrences. Because of that sorted list, every line could be n-gram transformed, normalized by the number of n-grams and rated. Line rating is performed by summing all indexes (in ascending sorted n-gram sequence) where n-gram items from the line can be found.

4. Results and discussion

Using preprocessing techniques described in Subsection 3.1.1. we obtained results showing significant reduction of conversations as shown in Table 1. When it comes to testing the classifiers, we have conducted two tests. The first test was to decide which classifier was the best for classifying conversations. Since "Victim vs. Predator" classification depends on the first task, we did not want to miss-identify suspicious conversations, so we decided to prefer the recall to the precision. As a result, we chose an F_2 score for evaluating the suspicious conversations problem. Classifier results comparison for the first test are shown in Table 2. As a result, we chose SVM with linear kernel and TF-IDF weighting. N-gram range that worked best is from 1 to 4. After nested 5-fold was done, it appeared that the best hyperparameter is C = 2.

Similarly, the second "Victim vs. Predator" test showed that the best result was achieved by SVM with linear kernel. Evaluation is done using $F_{0.5}$ score. Features were extracted using TF-IDF weighting with N-gram range from 1 to 2 and additional features as described in Section 3.1.2.. Nested 5-fold optimized SVM's hyper-parameter and gave the result C = 2.

Finally, our results on the test dataset showed that for the first problem of identifying the sexual predator we had $F_{0.5}$ score of 0.883, which is shown in Table 4. However, our solution was much better for the second problem. When we passed all predator lines to the evaluator it returned F_3 score of 0.499. Table 5 shows the complete list of the contestants' models ordered by F_3 score.

5. Conclusion

As we can see, there are some words and features that discern sexual predators from regular users, whether they are talking in non-sexual sense or even flirting. It is advisable to prefilter the corpus because it is less likely that sexual predators will approach the person it the conversation is not one-on-one. Using SVM with linear kernel and TF-IDF weighting is a very good idea since there are words that predators use more often than other users. Additional features, such as the number of conversations started, number

Table 2: Various classifier-vectorizer results for suspicious conversations detection.

Vectorizer	BoW	BoW	BoW	BoW	TF-IDF	TF-IDF	TF-IDF	TF-IDF
n-gram range	1 to 1	1 to 2	1 to 3	1 to 4	1 to 1	1 to 2	1 to 3	1 to 4
SVM - linear	0.920	0.914	0.892	0.872	0.964	0.964	0.969	**0.972**
K-nearest neighbors (43)	0.468	0.354	0.261	0.191	0.821	0.771	0.784	0.787
K-nearest neighbors (7)	0.629	0.509	0.391	0.303	0.802	0.695	0.653	0.631
Multi-layer Perceptron	0.953	0.887	0.644	0.931	0.956	0.920	0.592	0.922
Random forest (10)	0.487	0.410	0.304	0.235	0.447	0.387	0.312	0.304
Random forest (30)	0.556	0.428	0.299	0.235	0.483	0.383	0.316	0.283

Table 3: Various classifier-vectorizer results for Sexual Predator identification.

Vectorizer	BoW	BoW	BoW	BoW	TF-IDF	TF-IDF	TF-IDF	TF-IDF
n-gram range	1 to 1	1 to 2	1 to 3	1 to 4	1 to 1	1 to 2	1 to 3	1 to 4
SVM - linear	0.817	0.914	0.776	0.783	0.881	**0.923**	0.825	0.849
K-nearest neighbors (7)	0.625	0.606	0.606	0.586	0.167	0.167	0.167	0.167
Multi-layer Perceptron	0.559	0.703	0.328	0.640	0.455	0.562	0.321	0.375
Random forest (10)	0.620	0.476	0.094	0.094	0.154	0.446	0.088	0.205
Random forest (30)	0.560	0.449	0.154	0.175	0.290	0.342	0.290	0.154

Table 4: Predator identification results.

Model	Precision	Recall	$F_{0.5}$ score
villatorotello	0.980	0.787	0.935
snider121	0.984	0.721	0.917
Our solution	**0.971**	**0.650**	**0.883**
parapar12	0.939	0.669	0.869
morris12	0.969	0.606	0.865
eriksson12	0.857	0.894	0.864
peersman12	0.894	0.598	0.814

Table 5: Predator lines identification results.

Model	Retrieved	Relevant	F_3 score
Our solution	**62024**	**6003**	**0.499**
grozea12	63290	5790	0.476
kontostathis	19535	3249	0.417
peersman12	4717	1688	0.268
sitarz12	4558	1486	0.236

of emoticons and question marks used, along with TF-IDF features are strong "victim vs. predator" separators.

However, like other PAN2012 contestants mentioned, we also express our concern that the given $F_{0.5}$ and F_3 scores might not be set well. If the system discovers potential predators and passes the list to the authorities, it would be better to have more potential predators in the list, than to be more precise. In that way, the system can let the real person decide whether it is a real or fake predator without losing the potential ones. Moreover, because the F_3 score prefers recall to such an extent, it makes no sense to discuss which lines are more important. Therefore, it is best to pass all predator lines to get a high recall.

Compared to other contestants, we achieved the third best result for the predator identification problem. In finding the most distinctive lines of predators' bad behavior we achieved the best result. Further classifier testing and hyperparameter optimization could lead to better performance. It is important to mention that adding additional behavioral features could also slightly improve the results.

6. Acknowledgement

We wish to express our sincere gratitude to Mr. Damir Sekulić for proofreading and making our paper more readable.

References

Giacomo Inches and Fabio Crestani. 2012. Overview of the international sexual predator identification competition at pan-2012. In *CLEF (Online working notes/labs/workshop)*, volume 30.

Javier Parapar, David E Losada, and Alvaro Barreiro. 2012. A learning-based approach for the identification of sexual predators in chat logs. In *CLEF (Online Working Notes/Labs/Workshop)*.

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.

Marius Popescu and Cristian Grozea. 2012. Kernel methods and string kernels for authorship analysis notebook for pan at clef 2012.

Esaú Villatoro-Tello, Antonio Juárez-González, Hugo Jair Escalante, Manuel Montes-y Gómez, and Luis Villasenor Pineda. 2012. A two-step approach for effective

detection of misbehaving users in chats. In *CLEF (Online Working Notes/Labs/Workshop)*.

Author Index