Data Science Coding Challenge: Twitter Sentiment Analysis

Vasco Fernandes

August 26, 2018

Contents

1 Introduction

Hate speech is an unfortunately common occurrence on the Internet. Often social media sites like Facebook and Twitter face the problem of identifying

and censoring problematic posts while weighing the right to freedom of speech. The importance of detecting and moderating hate speech is evident from the strong connection between hate speech and actual hate crimes.

Early identification of users promoting hate speech could enable outreach programs that attempt to prevent an escalation from speech to action. Sites such as Twitter and Facebook have been seeking to actively combat hate speech. In spite of these reasons, NLP research on hate speech has been very limited, primarily due to the lack of a general definition of hate speech, an analysis of its demographic influences, and an investigation of the most effective features.

2 Problem Statement

The objective of this project is to find the best classification model to classify tweets.

The objective of this task is to detect hate speech in tweets. For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets.

Formally, given a training sample of tweets and labels, where label '1' denotes the tweet is racist/sexist and label '0' denotes the tweet is not racist/sexist, your objective is to predict the labels on the test dataset.

3 Data Set

3.1 Train Set

The data set provided (https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/) is composed by two different files, the train set ("*train_E6oV3lV.csv*") and test set ("*test_tweets_anuFYb8.csv*"). The first file is composed by 31621 observations, with 3 columns ("id", "label" and "tweet"), with an uneven label (or class) distribution, as can be seen in Figure 1:

3.2 Test Set

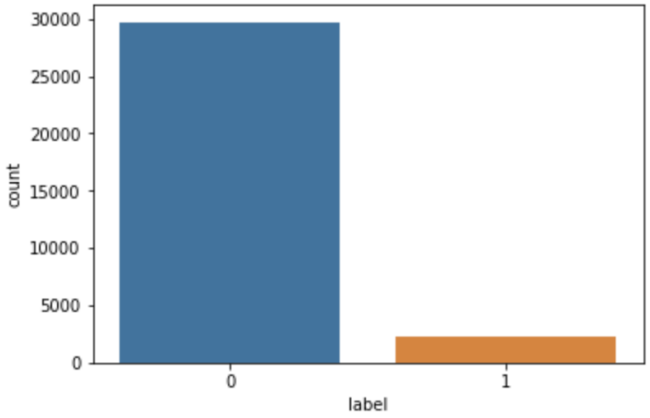
The test set is composed by 17197 observations, with only two columns ("id" and "tweet"). Given that this challenge is an open challenge, makes sense no "label" column, to make users submit their results.

4 Initial Feature Engineering

4.1 Pre-clean Tweet length

Given that in NLP tasks the most difficult challenge is the feature engineering part, and after applying algorithms out-of-box **after** cleaning and I was not able

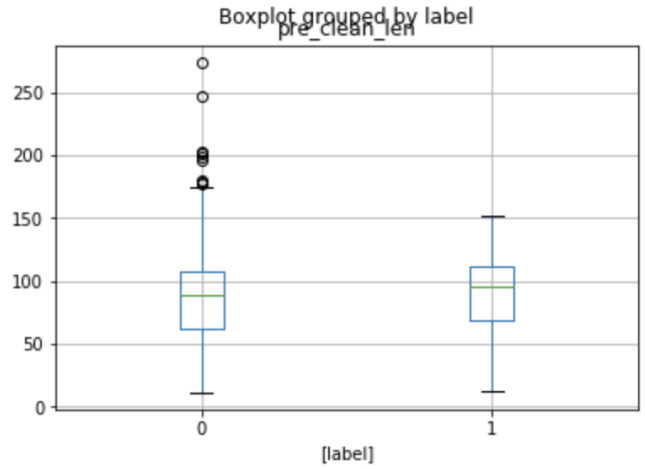
Figure 1: Class distribution



to produce good results, I tried to create my own features.

So the first hypothesis that I wanted to test was if one class of tweets has a statistically different pre-clean length that the other one. As we can see in Figure 2:

Figure 2: Pre-clean length

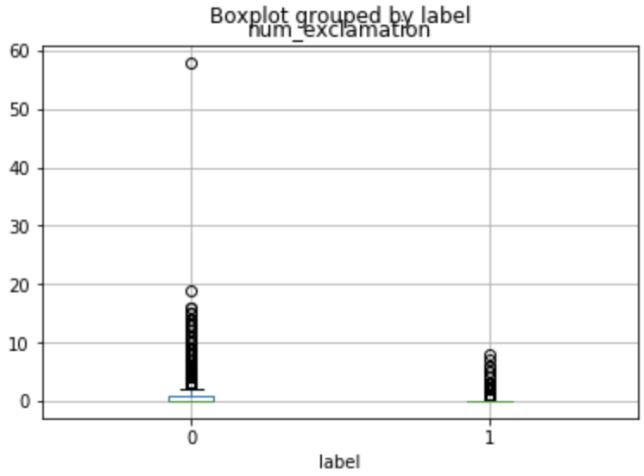


So, in terms of the median value, there is not a significant difference, but the 75^{th} percentile is somewhat deviated in class 0, comparing to class 1.

4.2 Number of exclamation marks

So following the same line of thought of the last subsection, I thought that exclamation marks could be a discriminative feature. The hypothesis was that hate-speech tweets had more exclamation marks than non-hate speech tweets. The boxplot for each class in terms of the number of number of exclamation points.

Figure 3: Number of Exclamation Marks



Again, there is not a significant difference between the two, but we can see, opposite to what I thought could be possible, the non-hate speech class has more exclamation marks.

4.3 Post-cleaning length

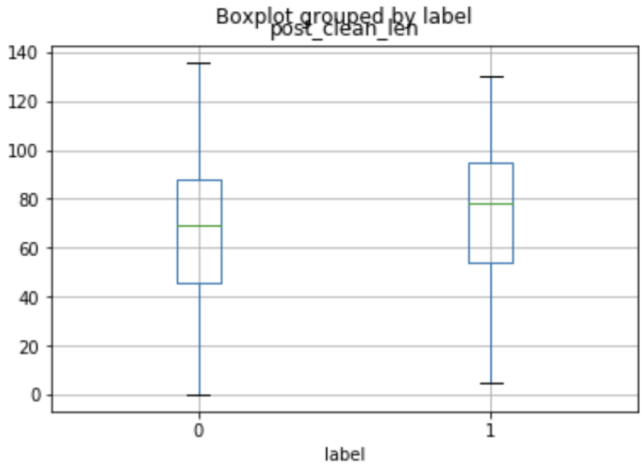
Skipping some steps, lets now check the post-cleaning length. Again, there is no statistically significant differences between one class and the other. This can be seen in Figure 4.

5 Data Cleaning

In this section, I will refer the cleaning steps that I took.

Given that the dataset already had user mentions anonymised, these mentions, in my opinion, did not had any predictive power. Given this, I wrote a function, called `remove_pattern(text, pattern)` that can remove any pattern of a *regex* expression. I applied this function to the data set column `tweet`, creating a new column, named `new_tweet`.

Figure 4: Post-cleaning length



The second step of the cleaning part was to lowercase each word of the column. This operation was applied to the `new_tweet` column.

The cleaning part of this project was also accomplished using another function `clean_tweet(text)`, that perform the following steps:

- Remove HTML characters.
- Encoding normalization.
- Check for common abbreviation in english and transform them to the long version (like "aren't" to "are not").
- Remove non-character symbols.
- Tokenize each tweet using the NLTK Tokenizer.

Then checked for missing values in all columns, that produced no results.

6 Exploratory Data Analysis

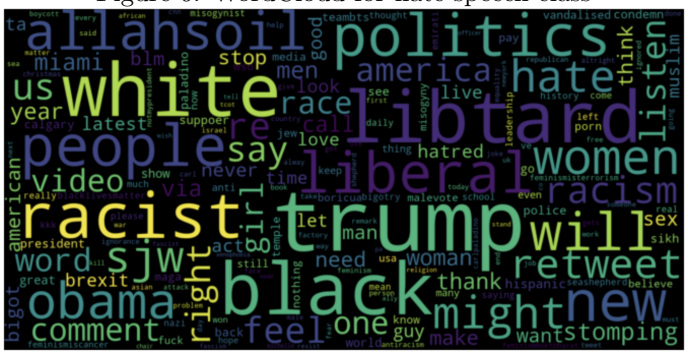
6.1 Word Cloud

For the positive class (non-hate speech), this is the word clouds after removing the stop words (Figure 5).

Now for the negative class (hate speech tweets), the word cloud can be seen in Figure 6.

Figure 5: WordCloud for non-hate speech class

Figure 6: WordCloud for hate speech class



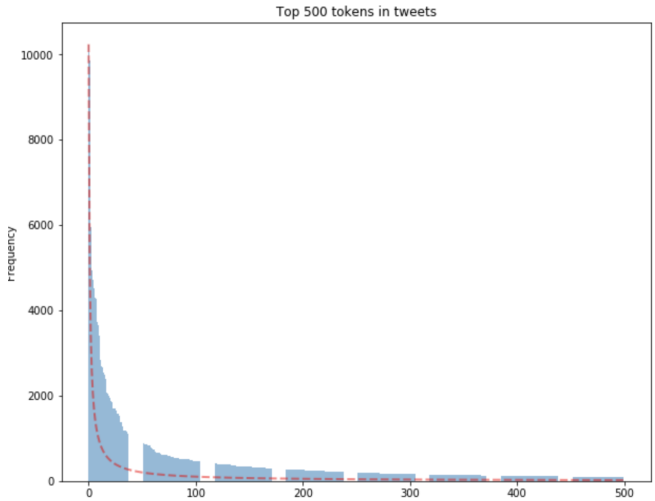
6.2 Zipf's Law

Zipfs Law states that a small number of words are used all the time, while the vast majority are used very rarely. There is nothing surprising about this, we know that we use some of the words very frequently, such as the, of, etc, and we rarely use the words like aardvark (aardvark is an animal species native to Africa). However, whats interesting is that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. Thus the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.

In Figure 7, the x-axis is the rank of the frequency from highest rank from left up to 500th rank to the right. Y-axis is the frequency observed in the corpus (in this case, our dataset). One thing to note is that the actual observations in most cases does not strictly follow Zipfs distribution, but rather follow a trend of near-Zipfian distribution.

Another way to look at this data is to plot the features at a log-scale. This can be seen in Figure 8.

Figure 7: Approximate Zipf Law



6.3 Token Frequency

Now let's go deeper into token frequency and test several metrics in order to see if we can extract meaningful information from this set of metrics.

Token count per class After removing the stop words, the first question was "how many times a certain token appears in one class?". But lets look at the frequency values of the minority class. Everything is pretty normal, right? (In terms of the ratio between the positives and negatives). What about words like "like", "just"? (Figure 9).

Most common tokens per class So I started to look for the most common tokens after removing the stop words. In Figure 10 and Figure 11, this can be seen.

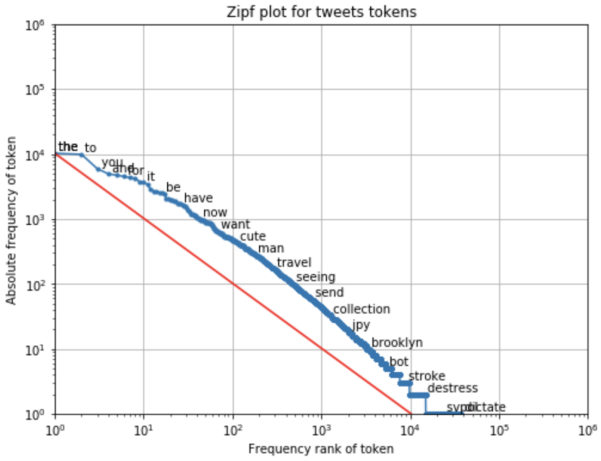
After this, I tried to plot the frequency of the positive and negative tokens in a scatter plot (Figure 12):

In the plot above, my goal was to find if there is some relationship between the frequency in relation to positive and negative tokens. The scale of the plot is not the same in the two axis, so one has to be careful when analyzing this results.

NOTE: each point represents a token!

NOTE 2: The tokens that are more meaningful are the ones in the 1st and 4th quarter. The point of no significant would be the points in the line y=x. A measure of meaningfulness would be the distance to this line!

Figure 8: Log-scaled approximate Zipf Law



Other Metrics Using the metrics for ranking tokens from this `https://www.youtube.com/watch?v=H7X9CA2pWKo` PyData Conference, lets explore what we can get out of frequency of each token. Intuitively, if a word appears more often in one class compared to another, this can be a good measure of how much the word is meaningful to characterise the class.

I call this the `pos_rate`, that basically is a measure of how much a token appears in the positive class related to the positive and negative classes.

So the metric above is not really useful, because, this rate can be really high if a word appears few times in the positive class and none on the negative class.

Given this, I used other metric, `pos_freq_pct`, that basically measures the frequency in class.

But since `pos_freq_pct` is just the frequency scaled over the total sum of the frequency, the rank of `pos_freq_pct` is exactly same as just the positive frequency.

So, and borrowing again from the metrics used in the PyData Conference in the ScatterText package, I wanted to combine the two metric above. However, the range of the two metric is different. Although they both range in the interval [0, 1] the frequency in-class ranges from [$\tilde{0}$.0072, 0] and the `pos_rate` metric from [0,1]. If I computed the average between the two, one of the metric would have much more weight in the final result than the other.

Since the harmonic mean of a list of numbers tends strongly toward the least elements of the list, it tends (compared to the arithmetic mean) to mitigate the impact of large outliers and aggravate the impact of small ones.

The harmonic mean rank seems like the same as `pos_freq_pct`. By calculating the harmonic mean, the impact of small value (in this case, `pos_freq_pct`) is too aggravated and ended up dominating the mean value. This is again exactly same as just the frequency value rank and doesnt provide a much meaningful

Figure 9: Token frequency per class

	positive	negative	total
love	2801	27	2828
day	2384	9	2393
happy	1695	12	1707
just	1289	79	1368
like	1041	139	1180
life	1169	7	1176
time	1128	22	1150
today	1081	14	1095
new	931	72	1003
thankful	952	0	952

result.

What we can try next is to get the CDF (Cumulative Distribution Function) (inspiration taken from here: `https://svn.spraakdata.gu.se/repos/richard/pub/statnlp2016_web/l3.pdf` value of both `pos_rate` and `pos_freq_pct`. CDF can be explained as distribution function of X, evaluated at x, is the probability that X will take a value less than or equal to x. By calculating CDF value, we can see where the value of either `pos_rate` or `pos_freq_pct` lies in the distribution in terms of cumulative manner.

So finally, we can see that there is a relationship between the positive and negative samples. Each data point represents a token, and tokens in the first quarter are the most significant in terms of "positivity", and the data points in the fourth quarter are the most significant in terms of "negativity".

7 Algorithms & Feature Engineering

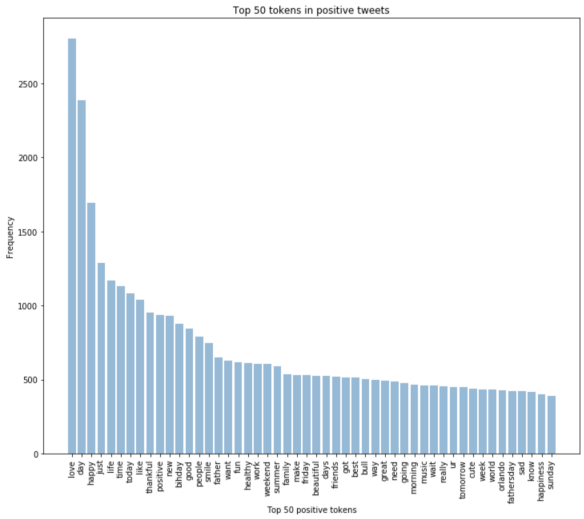
7.1 Logistic Regression

The first algorithm that I chose, given that I have worked before with logistic regression and due to being a quite interpretable and simple algorithm, I started the algorithmic part using Logistic Regression. In Figure 14, we can check the results using different `n_gram` ranges and different number of features extracted using TF-IDF feature extraction mechanism.

7.2 Algorithm Comparison

In this subsection, I experimented using different supervised algorithms with the number of features and `n_gram` range found to produced the best results

Figure 10: Token frequency per class



when applying the Logistic Regression algorithm. The results can be checked in Table 1.

Table 1: Algorithm Comparison

Algorithm	Hyperparameters Tuned	Feature Extraction Mechanism	N-Gram Range	Number of Features	Accuracy (Validation Set)	F1-Score (Validation Set)
LR	Yes	TF-IDF	1,2	90000	96.18%	75.88%
LinearSVC	Yes	TF-IDF	1,2	90000	96.32%	69.99%
LinearSVC with L1 FS	Yes	TF-IDF	1,2	90000	94.89%	66.53%
AdaBoostClassifier	Yes	TF-IDF	1,2	90000	94.82%	50.08%
MultinomialNB	No	TF-IDF	1,2	90000	94.31%	30.00%
BernoulliNB	No	TF-IDF	1,2	90000	94.27%	29.62%
RidgeClassifier	Yes	TF-IDF	1,2	90000	35.32%	11.66%
PassiveAggressiveClassifier	Yes	TF-IDF	1,2	90000	92.63%	51.73%
Perceptron	Yes	TF-IDF	1,2	90000	95.42%	60.12%
NearestCentroid	No	TF-IDF	1,2	90000	96.03%	70.12%

Doc2Vec In this part of this report, I will go through the experiments that I did using a different kind of feature extraction mechanism, called Doc2Vec. Given that this method learns a probabilistic (versus frequentist, like TF-IDF) vectorized representation of words, and this learning part is of unsupervised nature, the test data was used in this part. In Table 2, the results of different techniques can be observed and compared.

7.3 Deep Learning

The results when using various deep learning models can be seen in Table 3.

8 Discussion and Future Work

Figure 11: Token frequency per class

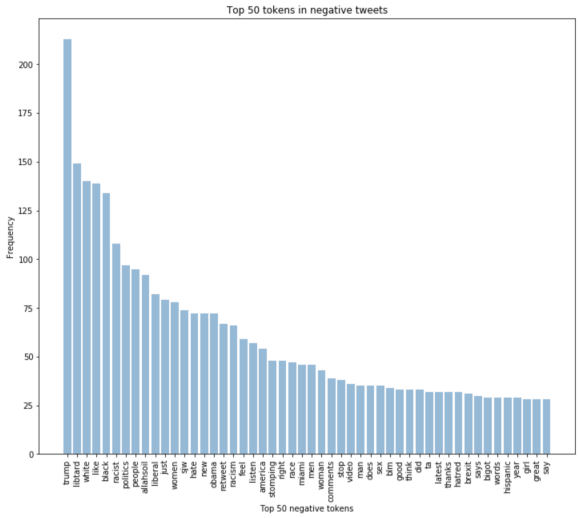


Table 2: Doc2Vec Results

Algorithm	Doc2Vec Technique	Accuracy (Validation Set)	F1-Score (Validation Set)
Logistic Regression	DBOW	80.15%	35.87%
Logistic Regression	DMC	86.01%	29.08%
Logistic Regression	DMC+DBOW	91.53%	35.84%

Figure 12: Token frequency per class

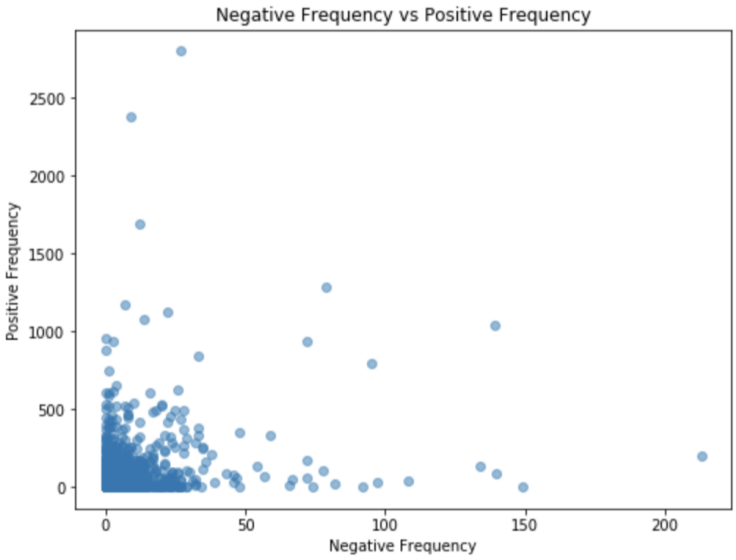


Figure 13: Token frequency per class

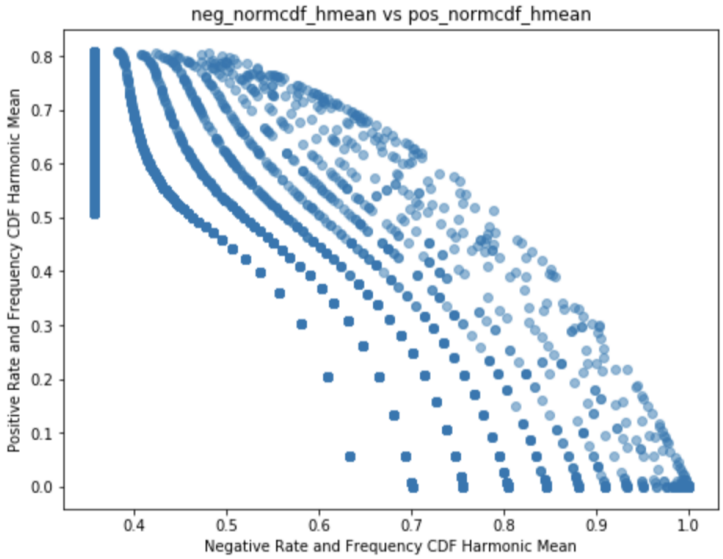


Figure 14: Logistic Regression results

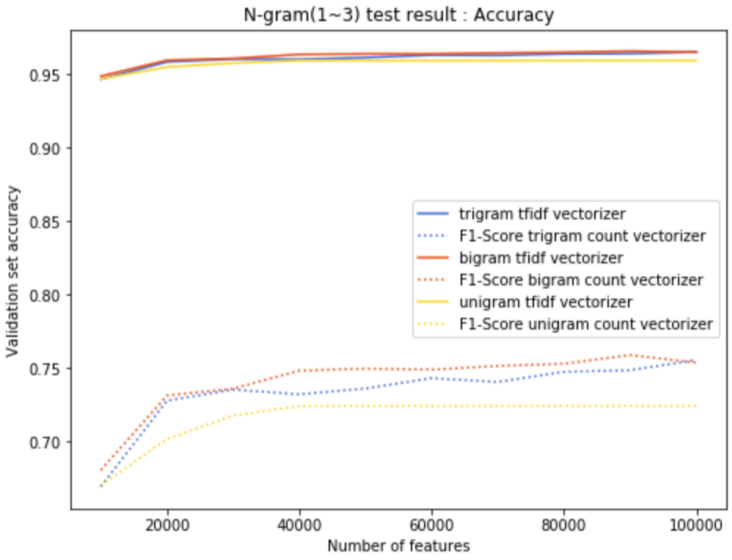


Table 3: Deep Learning results

Type	Number of Layers	Size of Layers	Epochs	Batch Size	Batch Shuffle	Batch Normalization	Optimizer	Dropout	Features	Accuracy (Validation Set)	F1-Score (Validation Set)
MLP	1	64	10	256	No	No	ADAM	No	TF-IDF	95.39%	67.05%
MLP	2	128/64	10	256	No	No	ADAM	No	TF-IDF	95.71%	65.91%
MLP	1	64	10	32	Yes	No	ADAM	No	TF-IDF	95.10%	4.86%
MLP	1	64	10	32	No	Yes	ADAM	No	TF-IDF	95.12%	4.86%
MLP	1	64	10	32	Yes	Yes	ADAM	Yes / 0.2	TF-IDF	95.18%	4.61%
MLP	1	64	10	32	Yes	Yes	Custom ADAM Lr=0.005	No	TF-IDF	94.92%	4.77%
MLP	1	64	10	32	Yes	Yes	Custom ADAM Lr=0.01	No	TF-IDF	95.01%	4.54%
MLP	2	128/64	40	256	No	No	ADAM	No	Doc2Vec (DMC+DBOW)	92.55%	41.47%
MLP	2	128/64	40	256	No	No	ADAM	No	Doc2Vec (DMC+DBOW)	93.35%	65.64%
MLP + Embeddings	2	256	5	256	No	No	ADAM	No	Word2Vec (DBOW)	95.25%	65.14%
MLP + Embeddings	3	256/256	10	256	No	No	ADAM	No	Word2Vec (DBOW)	96.02%	69.95%
CNN + Embeddings	8	N/A	10	512	No	No	ADAM	Yes / 0.2	Word2Vec (DBOW)	96.77%	74.88%