Text Analytics
2nd Assignment

Galanos Vasileios - p3351902

January 19, 2021

This report is one of the two deliverables which we provide as a solution to the 2nd Assignment for the course of Text Analytics of the MSc in Data Science at Athens University of Economics and Business. Accompanied with this file comes also the following link.

- Enter google colab link Google Colab Link.

Here, you can find the code written for the assignment. It is hosted in Github and Google Drive and can be opened through Google Colab. Below we provide an overview of what we have created, explanations for methods and patterns used, descriptions and opinions on tools, algorithms and practices.

Exercise 17- Sentiment classifier

For the purpose of this excercise we have implemented a sentiment classifier for the tweet texts extracted from the Twitter social network. The dataset we used is named "Sentiment140" and contains 1.6 million tweets.

Data Loading and Cleaning

We start by loading all the libraries we will later use in our code and then we download our dataset from the following link provided by the Stanford University.

- Sentiment140 Link.

Moving on, we unzip our dataset and load the data to a dataframe choosing "ISO-8859-1" for decoding as there were major failures when using Unicode, or "UTF-8" . After having a closer look at our data, we drop the extra columns that are not usefull for our assignment, as we only need the "text" and "sentiment" columns. Our dataset uses the "sentiment" column to denote the sentiment of the tweet, using "0" for negative, "2" for neutral and "4" for positive sentiment. We then, print the count of the column "sentiment " and check how many values are there from each category. As we can see, in our case we only have negative and positive values equally divided.

Next, we create a function (*tweet_cleaner*) cleaning our text from characters that are not usefull for the purposes of our assignment. The method is mainly based on regular expressions which help us achieve our goal. Some examples of cases and characters we want to remove are: references to other users via @ character, links to websites, single character words, hashtags, number and finally multiple spaces. We also occupy ourselfs with cases of html tags left in our texts and we care about removing faulty characters which might have been the outcome of the decoding process of the text. Finally, we lematize each word to merge multiple versions of the same words into one.

Then, we shuffle our data because, we have noticed that the positive sentiment tweets are placed lower on the dataset and we don't want that messing with our outcome [1]. Then, as we only have two values in our sentiment column, we map the "4" sentiment value (meaning "positive") to "1" in order to make our case looking binary. In the following code block, using the dataset we have manipulated until this point, we split it into three different sets:

- The largest set, will be our training set, consisting of the **70%** of the dataset.
- The next part, will be the development set, which will be used for hyper-parameter tuning and consists of the **20%** of the dataset.
- The final part, will be our test set consisting of the remaining **10%** of our dataset.

[1] shuffling the sentences makes the training more unbiased

Some statistics on the dataset

In order to understand our dataset better, we continue with extracting some statistics from our dataset. First, we get the document length per sentiment and as we can see the negative sentiment category has 1 more word on avergage, compared to the positive category. That may, or may not play a significant role in identifying the sentiment of the text.

Average word count per class

Class 0	13.87
Class 1	12.88

Next, we print the total number of documents in our subset of the initial dataset. [2]

Having seen the number of total documents interesting would be to see the size of each dataset. We can see the corresponding sizes below. Of course that process is only made, to verify our calculations when splitting our dataset.

Sizes for each set

Training set	70000
Development set	19999
Test set	9999

Features: Selection and vectors

Convert a collection of raw documents to a matrix of TF-IDF features. For this part of the exercise, we want to create feature vectors from our raw text documents and so we use TfidfVectorizer. We set the parameters of the TfidfVectorizer to use english stop words and use 5000 features at max and then use it, on all 3 of our datasets (training, development and test sets). We provide more information on feature selection at the corresponding appendix.

[2]For the purpose of this exercise we did not use the whole dataset, consistinf of 1.6 million tweets. Instead, we used a subset of 100000 tweets, which we believe is sufficient enough for the purpose of this exercise

Classification metrics

We created our own function (*get_scores*) that calculates the following metrics, for user-defined classification thresholds as well as for all the classes:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{F}_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$ [3]

Similarly we use macro-averaging to get a more robust estimate for all the classes

$$\text{Macro - Precision} = \frac{1}{n}\sum_{i=1}^{n} \text{Precision}_i$$

$$\text{Macro - Recall} = \frac{1}{n}\sum_{i=1}^{n} \text{Recall}_i$$

$$\text{Macro - F}_1 = \frac{1}{n}\sum_{i=1}^{n} \text{F1}_i$$

Classifiers

Moving on to the main part of the assignment, we will now try some classifiers to see how well, we can classify our tweets to the right sentiment. To find how well our classifiers have done on we first have to set a baseline , which helps us measure our performance later.

We choose as our baseline classifier, the DummyClassifier from sklearn, which we parameterize to always return the most frequent result. In order to make our code more streamline we create a pipeline which starts with a TfidfVectorizer and then uses the outcomes from the vectorizer on the DummyClassifier. We fit our training data on the classifier and then utilizing the methods we have previously created, we get our results. Comparing the results, we can see that there is no significant difference between the sets and that our macro-average f1 score is settled at 0.33 value.

All the scores presented bellow are measured for a classificiton threshold $t = 0.5$:

[3] We checked whenever the denominator was zero, and set the entire score to zero, in order to avoid arithmetic errors

Baseline classifier

Training set			
	Precision	**Recall**	**F_1**
Class 0	0	0	0
Class 1	0.502	1	0.668
Macro	0.251	0.5	0.334

Development set			
	Precision	**Recall**	**F_1**
Class 0	0	0	0
Class 1	0.495	1	0.663
Macro	0.248	0.5	0.331

Test set			
	Precision	**Recall**	**F_1**
Class 0	0	0	0
Class 1	0.498	1	0.665
Macro	0.249	0.5	0.332

For our next classifier we chose the SGDClassifier from sklearn. This is a set of linear classifiers with SGD training. For the loss parameter of the classifier we chose "log" which gives logistic regression and so in total we have basically Logistic regression as the classifier, but using SGD (Stohastic Gradient Descent) training. At this point, we could, either try our classifier and get its results, or we could use first GridSearchCV to find the best hyperparameters for our classifier and then find our results. We chose the second path, but in order to speed up the process we used RandomizedSearchCV instead, for which in contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. So, using RandomizedSearchCV and having created a pipeline using a TfidfVectorizer and the SGDClassifier, we find the best parameters from a concatenation of the training and the development set. Then, based on these best parameters we train our model and again utilizing our method, we get the scores of the model on all three of our sets. Again, as we can see our results don't differ that much from set to set. On the test set, which is the most interesting one, we get a decent score of 0.75 for the f1 using macro-average.

Logistic Regression classifier

Training set			
	Precision	**Recall**	**F$_1$**
Class 0	0.779	0.751	0.764
Class 1	0.761	0.788	0.774
Macro	0.77	0.769	0.769

Development set			
	Precision	**Recall**	**F$_1$**
Class 0	0.764	0.735	0.749
Class 1	0.74	0.769	0.754
Macro	0.752	0.752	0.751

Test set			
	Precision	**Recall**	**F$_1$**
Class 0	0.766	0.732	0.748
Class 1	0.741	0.774	0.757
Macro	0.753	0.753	0.753

Next, we thought that it would be usefull to use the RandomForestClassifier from sklearn. Using again the same process, we create a pipeline consisting of the TfidfVectorizer and the RandomForestClassifier. Then we use again RandomizedSearchCV to find the best hyperparameters and having them found, we train our model on the training set and then utilizing our method we get the corresponding results on each of our sets. The results we get are almost double in value on all sets from our baseline but have value around 0.65 for the macro-averaged f1 score, they are not better from the ones we got from the SGDClassifier.

Random Forest classifier

Training set			
	Precision	**Recall**	$\mathbf{F}_1$
Class 0	0.801	0.5	0.616
Class 1	0.639	0.877	0.739
Macro	0.72	0.689	0.678

Development set			
	Precision	**Recall**	$\mathbf{F}_1$
Class 0	0.79	0.492	0.606
Class 1	0.626	0.867	0.727
Macro	0.708	0.679	0.667

Test set			
	Precision	**Recall**	$\mathbf{F}_1$
Class 0	0.784	0.478	0.594
Class 1	0.622	0.868	0.725
Macro	0.703	0.673	0.659

As the fourth and last classifier we chose to use the Naive Bayes classifer via MultinomialNB model from sklearn. For this classifier we will again use the same process. We start by creating a pipeline consisting of the TfidfVectorizer and the MultinomialNB model. Then we use again RandomizedSearchCV to find the best hyperparameters and having them found, we train our model on the training set and then utilizing our method we get the corresponding results on each of our sets. The results we get are again by far better than the baseline we have set and as the value of 0.742 is really close to the 0.75 macro-average f1 value we got on the test set for the SGDClassifier, we cannot really say that this small difference in not based on the suffling of the dataset or another random factor. Both these classifier seem tho have decent results in our cases.

Naive Bayes classifier

Training set			
	Precision	**Recall**	**F_1**
Class 0	0.776	0.767	0.772
Class 1	0.771	0.781	0.776
Macro	0.774	0.774	0.774

Development set			
	Precision	**Recall**	**F_1**
Class 0	0.756	0.743	0.749
Class 1	0.743	0.756	0.749
Macro	0.749	0.749	0.749

Test set			
	Precision	**Recall**	**F_1**
Class 0	0.755	0.734	0.744
Class 1	0.739	0.76	0.749
Macro	0.747	0.747	0.746

Learning curves

Moving on to the learning curves from the Naive Bayes and Logistic Regression model we can comment on the following. The curve of the train score reduces as the score of the development set increases. This is only natural and the two curves seem to converge at a certain point but then keep finally a steady distance between them. The distance between the two curves could be corrected, if we had more training data and thus we can assume that our models needs better fitting. The plots containing the curves from the two models are diplayed below.

Figure 1: Learning Curves (Naive Bayes)

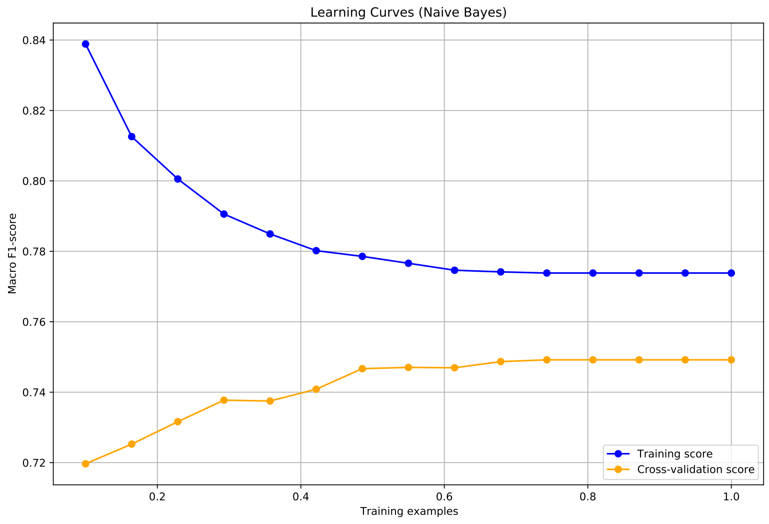
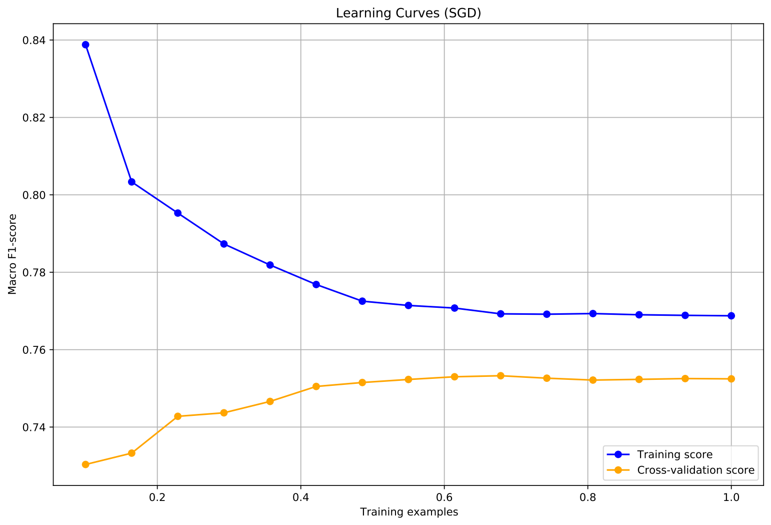
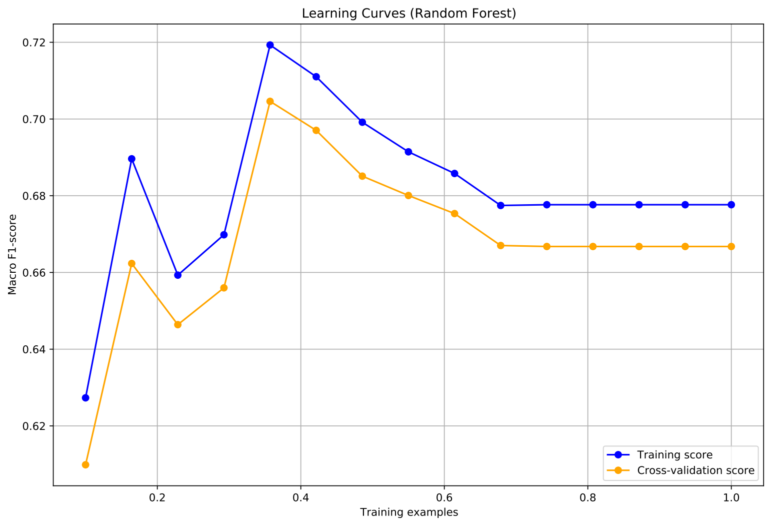


Figure 2: Learning Curves: Logistic Regression



On the Random Forest Classifier learning curves, we can see that the training score reduces dramatically and this path is also followed by the development set score. These is a massive drop at the half of the graph and then the two scores grow again alongside and stabilise at a certain value keeping a steady distance between them. This unsteady behaviour of the two curves could be expressed in many ways. One way could be that, the Train Dataset is unrepresentative, meaning that the training dataset does not provide sufficient information to learn the problem, relative to the validation dataset used to evaluate it. Another way to express this situation would be that the Validation Dataset is unrepresentative, meaning that the validation dataset does not provide sufficient information to evaluate the ability of the model to generalize.

Figure 3: Learning Curves: Random Forest Classifier



Precision-recall curves

Using our own function *get_scores*, we implemented plot_precision_recall_curve that, given a list of thresholds, calculates **precision** and **scores** and stores them to lists. Then we calculate the area under curve and plot a figure with those metrics for each threshold value.

Figure 4: Precision Recall Curves: Naive Bayes

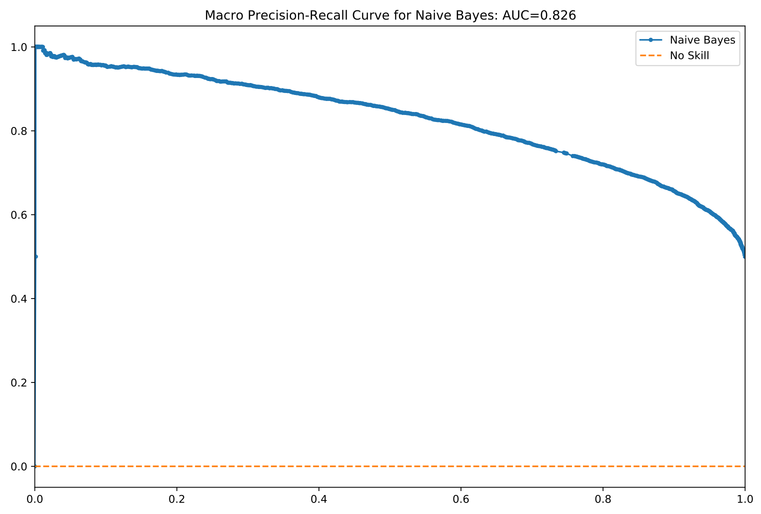


Figure 5: Precision Recall Curves: Logistic Regression

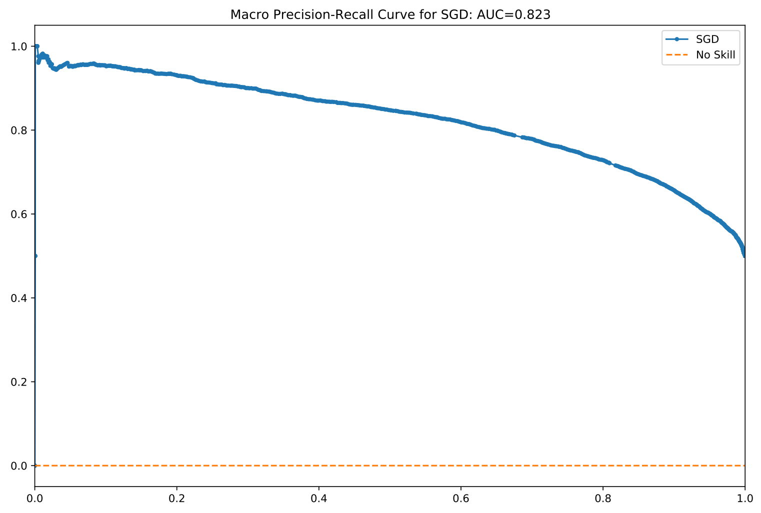
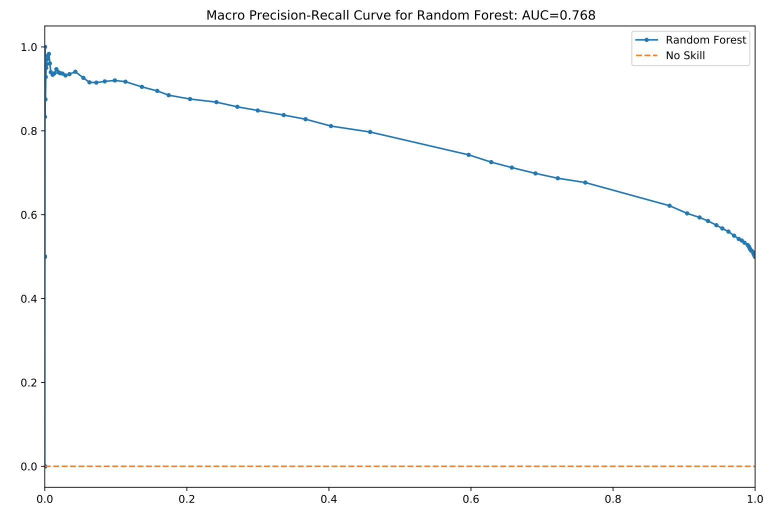


Figure 6: Precision Recall Curves: Random Forest Classifier



Interestingly, in terms of **AUC** (area under curve), the **Naive-Bayes classifier** is the best among the others, although it scored slightly worse in the previous sections metrics. This proves that these kind of figures have great value when one evaluates different classifiers. Just because on classifier outscored the other or a certain threshold, doesn't mean that it is the overall best classifier.

Bootstrap statistical significance tests

In order to derive a safe conclusion on which classifier performs outperforms the others, we use **Bootstrap Hypothesis Tests**. Let A the best classifier in terms of some metric on a test set and B another classifier that we want to compare it with. The main idea here is that we assume the null hypothesis (H_0) is that the difference between the two classifiers, in the chosen metric, should be zero. The alternative hypothesis (H_1), is that indeed, there is a significant difference between the evaluation scores not a random fluke.
Let $\delta(x) = \text{Macro-F1}_\text{A} - \text{Macro-F1}_\text{B}$, be the difference calculated on all the test set.

We state the following hypotheses as the null and alternative one:

$$H_0 : \quad \delta(x) = 0 \quad VS \quad H_1 : \quad \delta(x) > 0$$

Here, we define $\delta(x_i^*)$ as the differrence between the classifiers on a simulated dataset, sampled from the test set with replacement. We calculate the p-value as $p - value \approx \frac{s}{b}$ where s are the times that $\delta(x_i^*) > 2\delta(x)$, as the algorithm of Berg & Kirkpatrick states.

if $p - value < 0.01$, we have strong statistical evidence to reject the Null Hypothesis, thus embracing the alternative hypothesis as the correct one (i.e. accept that A's victory was real and not just a random fluke).

Applying this test using our **Logistic Regression** as our best classifier and comparing it to all the others we got the following results:

Logistic Regression VS			
	Baseline	**Naive Bayes**	**Random Forest**
p-value	0	0.266	0

As we saw in the results, we reject the null hypothesis in two out of three cases. Thus, the **Logistic Regression** classifier has significant difference with the **Baseline classifier** and **Random Forest classifier** and does not have a significant difference with the **Naive Bayes classifier**

Appendix A: Feature Selection

In this part, we perform dimensionality reduction to the tf-idf sparse matrix using Truncated SVD. We created a middle step in the pipeline and the training features reduced from 5000 to 500 before the training of Logistic Regression classifier. As we can see from the scores bellow, we can notice a slight decrease in the scores of all datasets.

Logistic Regression classifier with SVD

Training set			
	Precision	**Recall**	**F$_1$**
Class 0	0.733	0.734	0.734
Class 1	0.736	0.734	0.735
Macro	0.734	0.734	0.734

Development set			
	Precision	**Recall**	**F$_1$**
Class 0	0.733	0.737	0.735
Class 1	0.731	0.727	0.729
Macro	0.732	0.732	0.732

Test set			
	Precision	**Recall**	**F$_1$**
Class 0	0.734	0.735	0.734
Class 1	0.732	0.731	0.732
Macro	0.733	0.733	0.733

Moving on the learning curve, we can observe a decrease at the begining both on the training and development score following by a slight increase in the next samples. It is noticeable that the AUC (area under curve) is 0.80 comparing to 0.827 in the first case.

The Precision and Recall curve produces a figure with $AUC = 0.801$, which is not that dramatic, considering that we decreased our feature space a lot.

Figure 7: Learning Curve: SGD Classifier with SVD

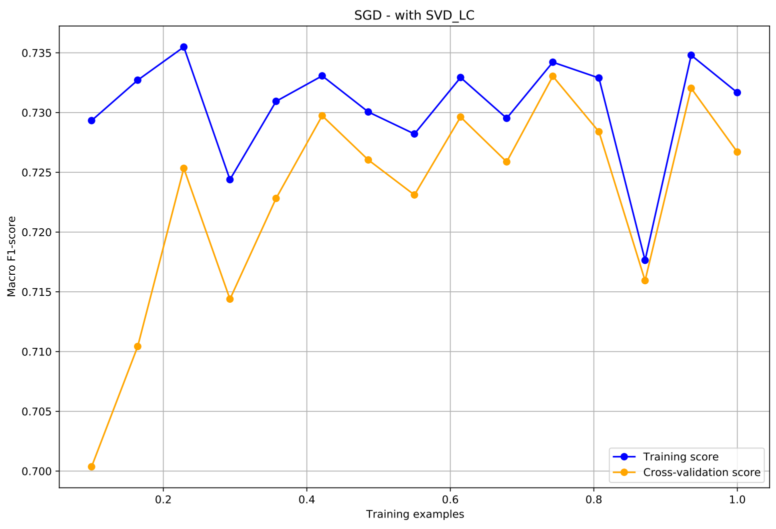
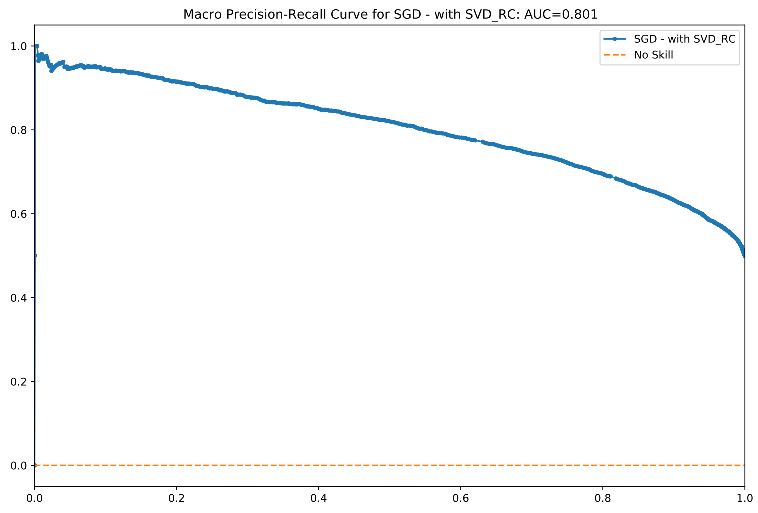


Figure 8: Precision Recall Curve: SGD Classifier with SVD



Appendix B: Word Embeddings

Here we present our work using Word Embeddings for for feature selections. We use centroids of pre-trained word via Word2vec provided by Google and train our classifiers using these centroids. The model contains 300-dimensional vectors for 3 million word and phases, and it has been trained on part of Google News dataset. Each tweet represented by averaging the word embeddings of all words. We represent the following scores for each classifier.

Logistic Regression classifier with centroids of word embeddings

Training set			
	Precision	**Recall**	**F_1**
Class 0	0.72	0.783	0.75
Class 1	0.759	0.692	0.724
Macro	0.739	0.737	0.737

Development set			
	Precision	**Recall**	**F_1**
Class 0	0.722	0.785	0.752
Class 1	0.761	0.694	0.726
Macro	0.742	0.74	0.739

Test set			
	Precision	**Recall**	**F_1**
Class 0	0.717	0.782	0.748
Class 1	0.77	0.702	0.735
Macro	0.743	0.742	0.741