Text Sentiment Classification

Yamane El-Zein, Camila Andrea González Williamson, Luis Emmanuel Medina Ríos,
Department of Computer Science, EPF Lausanne, Switzerland
CLY - *Kaggle Team*

***Abstract*—In this report, we discuss our findings for the PCML project on text sentiment classification. Our work involves three main steps: a brief analysis of the text data, preprocessing of the text data, and testing different variations of text classification methods. These methods include the bag of words method, two word embeddings techniques, namely Word2Vec and GloVe, and finally the FastText method. We describe in detail all tested methods and compare their results. In the end, we find that an ensemble of 200 FastText classifiers produces the best results.**

I. Introduction

The goal of this project is to come up with a method to predict the sentiment that a tweet conveys. Today, posts on social media, such as tweets, can hold valuable information and opinions. This information can be of interest to companies, political analysts, and marketers among others. Knowing the sentiment that tweets convey can be helpful in extracting popular opinions about various topics from a large dataset of tweets. Therefore, text sentiment classification is one of the most popular machine learning problems of our time. In this project, we perform classification on a dataset of tweets to predict whether a tweet message used to contain a negative (":(") or positive (":)") smiley face, by only taking the sequence of words in the tweet into consideration. We test several methods and compare the resulting accuracies of their predictions, in order to select the best one for text sentiment classification on our data. This report is organized as follows. The first section deals with the text analysis stage of our work. The second section summarizes our methods and result. The third section is a discussion where we compare and contrast the different methods, and explain the rationale behind the use of every one of them.

II. Text analysis

The training sample consists of 1250000 negative tweets and 1250000 positive tweets, contained in the files 'train_neg_full.txt' and 'train_pos_full.txt', respectively. In this way, the present classification problem deals with only two classes. In addition, we are given two subsets of the training data (positive and negative tweets) each one containing 100000 tweets. The testing sample consists of 10000 tweets, with an associated ID, for which we need to predict the sentiment (either positive or negative).

Most of the tweets are in English, and they only contain lower case characters. By taking a closer look at our data, we can note the use of contractions, abbreviations, numbers, and punctuation. Although the labeling symbols ':)' and ':(' have been removed from the data sets, we can still observe symbols such as ':p' and ':/'. Moreover, there are tweet specific words such as <user>, <url>, rt and Hashtags (words or combinations of words starting by a # symbol). By performing an analysis of the number of word occurrences in each of the positive and negative training sets, we can see that the 30 most common words are similar in both sets. Figures 1 and 2 present the word counts of most common words in both datasets.

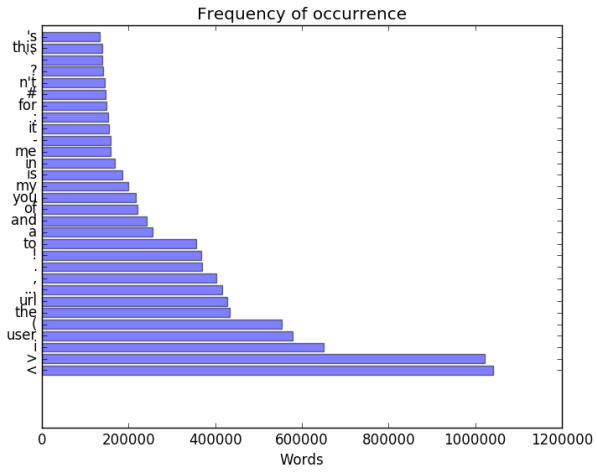

Figure 1: Count of common words in the negative training data set

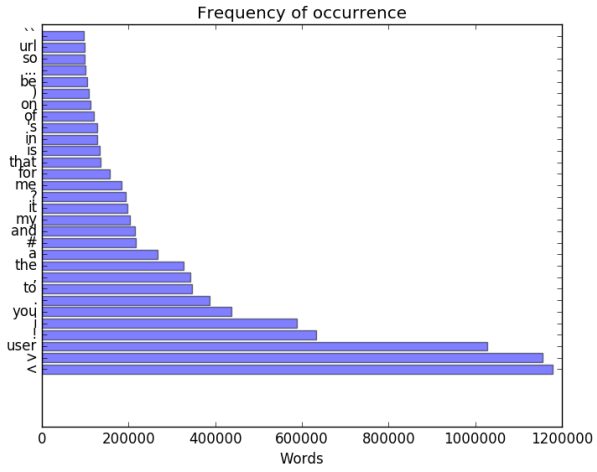

Figure 2: Count of common words in the positive training data set

III. Methodology and results

We experiment several text representation methods, combined with different classifiers. For text representation, we

examine GloVe, Word2Vec and the Bag of Words model. Given the different vector representations of the tweets, we experiment with logistic regression, SVM, neural networks and random forests for classification. We also examine FastText, which inherently combines text representation with classification. Furthermore, in order to verify our findings during the feature analysis, we test different types of pre-processing with FastText, which is the fastest classifier. Finally, we explore an ensemble of classifiers.

A. Word Embeddings: GloVe and Word2Vec

1) GloVe: Our first approach is to use the sample code provided to create embeddings for the tweets in the subset of the training data (100000 tweets). As a first step, we construct a vocabulary set consisting of the words that appear at least 5 times in our training set. From this vocabulary, we generate a co-occurrence matrix, on which we then train the GloVe model. This yields a numerical vector representation for every word in the vocabulary. The vectors consist of 20 features. Then, for each tweet in our training set, we construct a feature representation of the tweet by averaging the word vectors of the words found in the tweet. We ignore words that do not belong to the vocabulary in the averaging process. Having generated embeddings for all tweets in the subset of the training data, we proceed to train classifiers on these embeddings. As a baseline, we use logistic regression, which is faster than other classifiers. On a local validation set, the logistic regression model results in an accuracy of 59.61%. We then train an SVM model on the same embeddings, which results in an accuracy of 59.98%. We also train several neural networks with various numbers and sizes of hidden layers. We obtain the best accuracy of 63.47% with a neural network with 3 hidden layers, each composed of 30 hidden neurons. In addition, we train a random forest classifier with 100 estimators, and obtain an accuracy of 65.3%. The best classifier in this case is therefore the random forest classifier. We then repeat the process with word embeddings consisting of 200 features instead of 20. Since the training of SVM was very slow with 20 features and did not significantly improve the accuracy, we only train the other classifiers. With logistic regression, the accuracy reaches 62.4% on the local validation set. With the same neural network used previously, the accuracy reaches 66.87%, while with the random forest classifier, the accuracy reaches 66.0%. With a larger number of features, the accuracies increases for all classifiers. The neural network becomes the best classifier.

Table I: Summary of accuracies on local validation set using the GloVe method for different classifiers using word embeddings with 20 and 200 features.

No. of features	Logistic regr.	SVM	NN	Random Forest
20	59.61%	59.98%	63.47%	65.3%
200	62.4%	-	66.87%	66.0%

2) Word2Vec: Since our accuracies with the GloVe model are low, we create different word embeddings using the Word2Vec model. The Word2Vec model is an unsupervised learning algorithm that is based on neural networks and is attempting to learn the relationship between words in an automatic way [5]. We use the Gensim python library to employ the Word2Vec [6] and we work again with the subset of the training data (100000 tweets). To get embeddings for the tweets in our training set, we average the word vectors for the words in the tweet, as we did with the GloVe method. We make use of our previous findings from GloVe to optimize the choice of classifier and the parameters. First, we choose the number of features in the word vectors to be 300. This is the maximum number of features we can use without running into memory problems in our testing environment. We train a neural network with 3 hidden layers of 30 neurons each, which is the classifier that results in the highest accuracy when used with GloVe. We start with a context window size of 5. We get an accuracy of 79.72% on a local validation set. We increase the context window size to 10. This increases our accuracy to 80.305%. We further increase our context window size to 15. This decreases our accuracy to 80.16%. We conclude that a window size of 10 gives the best results on our dataset . When we compare the results of the Word2Vec method with those of the GloVe method, we notice that Word2Vec yields a much higher accuracy. This improvement is in part related to the fact that we were able to generate more features with the Word2Vec model without running into memory problems, which improved the effectiveness of the classification by the neural network. Another reason why the accuracy is higher with Word2Vec is because the embeddings are more representative of the words than with GloVe.

B. Bag of Words with predefined vocabulary

We also experiment with the Bag of Words model. The idea behind the bag of words model is quite simple and can be summarized as creating a vocabulary of unique tokens and then constructing a feature vector for each tweet representing the frequency of the vocabulary words. We use the scikit-learn python library, more specifically the CountVectorizer class [4]. Again, with the subset of the training data, we use the maximum number of words possible in the vocabulary that does not cause memory problems to get the highest accuracy possible. For each tweet, we get a vector with 1500 features (length of the vocabulary). Each value in this vector is the number of occurrences of a specific vocabulary word in that tweet. We then train several classifiers on the generated vectors. With logistic regression, we obtain an accuracy of 78.47% on a local validation test set. Again, we experiment with different neural networks and select the one that gives the best accuracy. In this case, we choose a neural network with 3 hidden layers, each composed of 10 neurons, which gives an accuracy of 80.227%. Finally, we train a random forest classifier with 100 estimators, and get an accuracy of 79.9%. We do not train an SVM model due to the large number of features in the data, which would lead to very slow training. The neural network is the best classifier used with the Bag of Words model. However, all three classifiers give similar results. In an attempt to improve accuracy on the Bag of Words

model, we experiment with bigrams and trigrams. We again test the same classifiers. Using bigrams slightly improves the accuracy only when using the logistic regression classifier, while worsening the accuracy for all other classifiers. Using trigrams worsens the accuracy for all classifiers. A summary of the results is shown in Table II.

Table II: Summary of accuracies on local validation set using the bag of words method for different classifiers and n-gram sizes

n-grams	Logistic regression	Neural Network	Random Forest
1	78.47%	80.227%	79.9%
2	78.52%	79.4%	79.55%
3	78.42%	79.5%	79.54%

We get better results with the Bag of Words model than with GloVe for all classifiers. The results from Word2Vec and the Bag of Words model combined with a neural network classifier are similar.

C. Bag of words without predefined vocabulary

Seeking for a higher accuracy level, we examine the Bag of Words model implemented with the CountVectorizer class [4], but without specifying a vocabulary for limiting the number of possible features. In addition, we use the full data set for training (2500000 tweets) and we take into account both unigrams and bigrams. Using this configuration, CountVectorizer generates a total of 5758593 features. Due to the large number of features, we are only able to test logistic regression (we experience several memory problems both with Neural Networks and Random Forest). With this setting, we obtain an accuracy on the local validation set of 85.1%. Given the large improvement in accuracy we decide to continue exploring logistic regression. Therefore, we add a regularization constant C and we tune it though a 5-fold cross-validation technique using a subset consisting of 200000 tweets (half of them positive and half of them negative). The result, is presented in Figure 3.

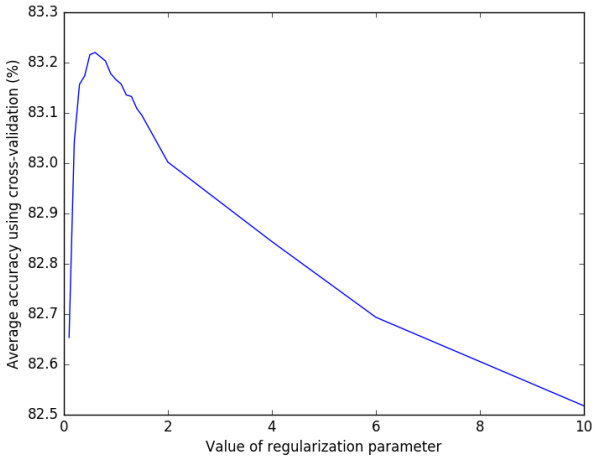

Figure 3: Tuning the regularization parameter for the Logistic Regression The best result is achieved with $C = 0.6$

In the plot, it can be seen that the optimal regularization constant (as defined by scikit-learn) is 0.6. Using this regularization constant, on the full data set increases our accuracy to 85.7% on a local validation set.

D. Fasttext

In line with our examination of methods, we focus our attention now on FastText. FastText is a library implemented by Facebook for word representation learning and text classification. It is based on the idea that *linear models with a rank constraint and a fast loss approximation can train on a billion words within ten minutes, while achieving performance on par with the state-of-the-art* [2]. Therefore, over the methods mentioned before, FastText has the advantage of being much faster, while achieving high accuracies. In particular, FastText creates word representations which are then averaged into a text representation. This text representation, is in turn fed to a linear classifier [2] with an specific loss function.

For the purpose of this project, we use FastText as a classifier. There are several parameters that can be adjusted when using the classifier. Based on research and testing, we decide to tune the following parameters by testing multiple combinations of them.

- lr: Learning rate. Its value is 0.1 by default
- loss: Loss function. The option by default is *Softmax* (sofmax function) but we can choose as well *ns* (negative sampling) or *hs* (hierarchical softmax)
- word_ngrams: Maximum length of word ngram. *Unigrams* by default

Some of the results of the parameters tuning are presented in Table III.

Table III: Parameters tunning for Fasttext

loss function	word_ngrams	lr	Accuracy
ns	1	0.022	82.67%
hs	1	0.022	82.71%
softmax	1	0.022	82.80%
ns	2	0.022	84.9%
hs	2	0.022	84.33%
softmax	2	0.022	84.82%
ns	3	0.022	85.09%
hs	3	0.022	84.14%
softmax	3	0.022	84.62%
ns	4	0.022	85.03%
hs	4	0.022	83.75%
softmax	4	0.022	84.43%

At the end, we chose to use the following parameters values: lr: 0.022, loss: ns, word_ngrams: 3, min_count: 4, bucket: 2000000, thread: 4

The reason is that they gave us the best accuracy without demanding a very high running time for each execution of FastText.

Now, taking advantage of the speed of FastText as a classifier, we tested different types of text cleaning in order to verify our findings during the text analysis. Some of the results are presented in Table IV.

Table IV: Text cleaning with Fasttext

Cleaning	Tweets w/o prediction	Accuracy
No cleaning	1	84.90%
Without stop words	3	83.33%
Without stop words and tags	3	83.18%
Without punctuation	2	82.29%
Without punct. & numbers	2	83.32%
Without single characters	2	83.53%
Combining all cleaning	9	80.58%

As we can see in Table IV we were obtaining better accuracy without making any kind of text processing, that is to say, the raw text.

E. Ensemble classifier

As a final step, and taking into account that we have two different methods giving a similar accuracy (Bag of Words with logistic regression and FastText), we decide to experiment with an ensemble of methods based on the majority voting. Specifically, what we aim to do is to aggregate the methods by averaging their predicted probabilities of being a negative tweet and then assigning the classification to each tweet according to these probabilities.

Furthermore, taking advantage of the fact that there is a source of randomness in FastText that we can not control, we construct the prediction for FastText as the ensemble of 50 and 200 FastText predictions with random combinations of the training set. The results of the accuracy obtained in the official testing data set for the best individual methods and the ensemble method are presented in Table V.

Table V: Summary of accuracies obtained on the actual test data set using ensembles of methods

Method	Accuracy
Single FastText	85.31%
Single Logistic regression	84.74%
Ensemble of 50 FastText classifiers	85.66%
Ensemble of 200 FastText classifiers	86.84%
Ensemble of logistic regression and 50 FastText classifiers	86.24%
Ensemble of logistic regression and 50 FastText classifiers	86.78%

As it can be seen in Table V, the ensemble of methods is clearly outperforming the individual methods. However, we can observe a very interesting behaviour: when we combine 50 classifiers of FastText with a single Logistic regression we outperform the ensemble of 50 FastText classifiers, but when we combine 200 FastText classifiers with a single Logistic regression, the ensemble of 200 FastText turns out to be the best classifier.

IV. Discussion

As it was evidenced in the previous section, in this project, we are having the challenge of finding the best numerical representation, i.e. the best features, for a set of tweets, which can be though as categorical variables. At the same time, we are having the challenge of finding the best machine learning algorithm, in terms of accuracy and computational requirements, to be fed with this set of features.

For the project, we are provided with a training sample consisting of 2500000 tweets and with a testing sample. It is important to mention that although having a very large data set for training gives the possibility of achieving a high level of accuracy, it constrains the problem in terms of memory requirements and running time of the machine learning algorithms.

As it was shown before, given the complexity of the problem, there are many approaches that can be taken. On one side, although the Word2Vec and Glove are more elaborated algorithms for the creation of a numerical representation for the tweets than the Bag-of-Words model, we obtained higher accuracies and lower running times with the latter. On the other hand, although SVM, Neural Networks and Random Forest Trees are much more sophisticated methods for classification, we found that it was more appropriate for this case to use logistic regression, as it is capable to handle a very large number of features, in a reasonable running time (less than one hour) and gives a good level of accuracy in comparison with the other methods. Continuing our line of experimentation we encountered the capabilities of FastText, which has a similar architecture to the continuous bag of words model (cbow) [3] for the text representation and uses a linear loss function for classification. We found that the algorithm, was able to provide a good solution to the problem at hand, both in terms of accuracy and running time.

Finally, based on the idea that combining different classifiers into a single one has a better generalization performance than each individual classifier alone [5], we made an ensemble of classifiers, getting the highest score over all the methods. This is in line with the fact that *ensembles are often much more accurate than the individual classifiers that make them up* [1] as they tend to reduce overfitting and variance and, at the same time, average out biases.

V. Conclusions and future work

In this project we had the challenge of predicting the sentiment (either positive or negative) of a set of tweets. For this purpose, we were given a large set of tweets (2500000) to train our model and a set of testing tweets. In the project we encountered the complexities of working with large training data sets causing very large execution times and/or memory problems. After trying different combinations of methods, we found that the best one was an ensemble of 200 Fasttext classifiers. Moreover, we found that removing stop word and other special characters, reduced the accuracy of our predictions. In the future, more advanced machine learning techniques such as long-short-term memory models (which consist of recurrent neural network) should be examined. Nevertheless, the power of making ensembles of different classifiers should be kept in mind.

REFERENCES

[1] T. G. Dietterich. "Ensemble Methods in Machine Learning". In: *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science (pp. 1-15). New York: Springer Verlag.* (2000).

[2] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *CoRR* abs/1607.01759 (2016). URL: http://arxiv.org/abs/1607.01759.

[3] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781 (2013). URL: http://arxiv.org/abs/1301.3781.

[4] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[5] Sebastian Raschka. *Python Machine Learning*. Birmingham, UK: Packt Publishing, 2015. ISBN: 1783555130.

[6] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.