Tweet Sentiment Classification

Beril Besbinar, Dimitrios Sarigiannis, Panayiotis Smeros
Team Name: *gLove is in the air*
Department of Computer Science, EPFL, Switzerland

***Abstract*—In this report, we present a comprehensive study of sentiment analysis on Twitter data, where the task is to predict the smiley to be positive or negative, given the tweet message. With a fully automated framework, we developed and experimented with the most powerful proposed solutions in the related literature, including text preprocessing, text representation, also known as feature extraction, and supervised classification techniques. Different combinations of these algorithms led to a better understanding of each component and exhausting test procedures resulted in a very high classification score on our final results.**

I. Introduction

Sentiment analysis, which is a common application of Natural Language Processing (NLP) methodologies, aims to obtain a sentiment score by quantifying qualitative data. Although a sub-task of binary classification can be attacked by simply using a dictionary of good and bad words, which represent +1 and -1 scores, respectively, and summing the scores of all words in a sentence/document for the final decision, disentangling the structure and relations of the words with each other may be quite challenging. Despite this difficulty, many companies, such as *Google* and *Facebook* and lots of research groups have an increasing interest on NLP tasks due to their applicability to daily life.

In this project, our task is to decide whether a tweet message, which can be composed of at most 140 characters, contains a positive or negative smiley. Although it may appear to be a standard binary text classification problem at first glance, it is slightly different and can be more difficult regarding the corpus, emojis that can be used within the message or hashtags, which are tweet tags that can be composed of multiple words used without a space between and '#' before.

In this report, you may find the brief explanation of possible preprocessing steps, text representation options and test results for a variety of classification algorithms, which is followed by a comparison and conclusion. Among them, two of our approaches achieved top-3 scores in Kaggle's competition[1], with the one employing a convolutional neural network to reach the 87.54% of correct predictions.

II. Data Description

The given dataset is composed of 2.5 million labeled and equally separated tweets that used to contain a positive :) and or negative :(smiley, which is supposed to be used for training, and another $10,000$ unlabeled tweets for testing. Given the smiley-filtered test tweets and considering only the remaining text, we are asked to classify them in positive and negative sentiments. The fact that training dataset is equally separated and the absense of a neutral sentiment made our problem easier.

[1]http://inclass.kaggle.com/c/epfml-text

On the other hand, there are some important properties of the dataset that should not be discarded. Twitter text is written by regular users in daily life, so in that informal language that is used for tweets, there are usually non existing words which nonetheless makes sense either in the morphological or in the pragmatic layer of a language. Moreover, there are plenty of characters that are not used for the formal written language, which is the basis of most pretrained models, but they usually indicate something meaningful in tweets, like punctuation or other types of emotions. Finally, the existence of compound words, syntactically incorrect sentences, different conjugations of the same word and many other similar cases are the problems to be attacked by preprocessing steps.

III. Preprocessing Steps

In this section, we describe the techniques that we use in order to process the raw tweets. After applying these techniques, we were able to emphasize writing styles and words that denote sentiment (e.g., emojis, repeating the last character of a word, etc.) as well as handling most of the spelling errors and slang words usage (e.g., with lemmatization and stemming).

Most of the implemented methods were inspired from the respective methods of *GloVe* [1] and are summarized as follows:

Contractions Expansion: In informal speech, which is widely used in social media, it is common to use contractions of words (e.g., *don't* instead of *do not*). This may result in misinterpreting the meaning of a phrase especially in the case of negations. For this reason, we implemented a method that expands these contractions.

Emojis Transformation: Emojis are widely used in tweets and they can alone give a good estimate of a post's sentiment. In order to handle the different writing styles of emojis (e.g., *:)*, *:-)* and *(:* are all smile faces), we transform all of them into some words that describe better the sentiment that they denote. Thus after emoji transformation, we have smile, lol, neutral and sad faces as well as hearts.

Emphasize Repeated Punctuation: Some punctuation marks, like exclamation point (!), give important clues about the sentiment especially when they are repeated within a tweet. We implemented a method in which we emphasize that repetition.

Emphasize Repeated Last Characters: Similar to the punctuation marks, the repetition of the last character of a word can also be a good indicator of the sentiment (e.g., *I am happyyyy*). We introduce a method that corrects the spelling of such words (e.g., converts *happyyyy* to *happy*) and adds a special tag to indicate the repetition.

Filter Numerical Expressions: Since numbers in general do not hold any special sentiment semantics, we replace all the numerical expressions with tags that stand for the existence of such expressions.

Split Hashtags: Hashtags are one or more words concatenated with the symbol #. Some of the tokens of the hashtags may hide useful information for the sentiment of a post (e.g., #lovemyjob). Splitting a hashtag into tokens is a difficult problem [2] since we may have multiple, ambiguous splits (e.g., #homestore can be split into either *home store* or *homes tore*). Having typos and using slang words makes it even more difficult. In order to solve this problem we used a dictionary with the most frequently used English words in descending order (according to Zipf's law) and tried to guess the correct split.

Emphasize Sentiment Words: Similar to emojis, some English words denote clearly an emotion (e.g., *anxiety* or *happiness*). In order to emphasize the existence of such words, we were advised by a lexicon with positive and negative words [3].

Part-Of-Speech Tagging: Part-Of-Speech tagging helps us to understand the structure of an input text in order to discover its most important words. Since posts in social media are informal, i.e., without obeying many grammatical rules, in most of the cases this method did not help.

Lemmatization and Stemming: In order to homogenize verbs being in different tenses, we implemented a method that applies lemmatization and stemming techniques to the words of the input tweets.

Stop-words Filtering: Stop-words are very common English words that can be found in almost every tweet and thus do not imply any sentiment (e.g., the articles *The*, *A* etc.). We implemented a method that removes the stop-words from tweets.

In addition to all these methods above, we have also experimented to use <user> and <url> tags as stopwords, since they appear in a very high frequency in the corpus, but we didn't observe any performance improvement for most of the algorithms. The single-character elimination did also not increased the accuracy.

IV. Representations of Tweets

In this section, we present all the algorithms that we investigated for converting our input tweets to numerical representations. These representations try to capture the semantic meaning of the tweets while having a constant size, which is independent of the words used in each tweet. Thus, letting N be the number of input tweets, we create an $N \times D$ matrix with each line representing a different tweet with D features.

Bag of Words Representation: Bag of Words [4] is a very common and straightforward numerical representation of text data. Given that we have a vocabulary V with finite size, we create an array of size $|V|$ with $1s$ in the position of the words belonging in this text and $0s$ everywhere else. Thus, in our case, we would have an $N \times |V|$ matrix representing our input tweets.

The problem with this representation is that $|V|$ can be arbitrary big especially if we do not handle appropriately slang and words with spelling mistakes, which can actually be solved by using the preprocessing methods that we proposed in Section III. Another problem of this representation is that it does not comprise any semantic information about the significance of each word in a tweet.

TF-IDF Representation: The significance of the words in tweets is captured by a more sophisticated representation, which uses the Term Frequency - Inverse Document Frequency (TF-IDF). TF-IDF [5] works by determining the relative frequency of a word in a specific document compared to the inverse proportion of that word over the entire document corpus. Words that are common in a single or small group of documents tend to have higher TF-IDF numbers than common words such as articles and prepositions.

As can be used as a standalone representation, this metric can also be used to weight the Bag of Words representation, in which case the $1s$ in $N \times |V|$ representation matrix are replaced by $TF-IDFs$.

N-grams Extension: In both of the above representations the $N \times |V|$ matrix can become even bigger if we add sequences of contiguous words in our vocabulary. These sequences are called *n-grams* with n being the maximum number of words in a sequence. For example, for bigrams, in the worst case the size of the vocabulary becomes $|V| + \binom{|V|}{2}$. As we will see in the experimental evaluation we tested representations with up to 8-grams.

Word Embeddings Representation: Word Embeddings (WE) [6] is a recently proposed representation that maps words to high dimensional vectors of real numbers. This representation tries to capture the multiple contexts in which a word can appear in a text. Hence, it guarantees that words that appear in similar context will have vectors that are very close each other.

The advantage of WE compared to the other aforementioned representations is that each word is represented by a fixed-size vector regardless of the used vocabulary. In tweet classification problem, given a of size $|V|$, tweets can be represented by a $|V| \times D$ matrix, where D is the dimension of our vectors. Depending on the vocabulary, we have different options for WE methods, which are listed below.

1) GloVe Training Methods: GloVe [1] has an open-source implementation of related training methods. As a first option, we used the given training tweet dataset to obtain the representation matrix using both the given implementation and *glove-python*[2], which is the python implementation of the original algorithm that uses *SGD* for training.

2) Pre-trained GloVe Embeddings: As another option, we also employed the pre-trained set of Word Embeddings for tweets, published by GloVe[3]. These vectors were trained with 2 billions tweets containing 1.2 millions words. Regarding the one order of magnitude difference in terms of the amount of training data, their corpus is observed to cover a high percentage of the words used in our given train set.

3) Hybrid Method: Since not all of the words in our training dataset were found in the pre-trained Word Embeddings, we decided to train our own vectors for these remaining words and append them to the pre-trained ones.

From Word Embeddings to Tweet Embeddings: All aforementioned representation methods enable us to construct vector representations for all the words in the vocabulary used by the tweets ($|V| \times D$ matrix). Since tweets contain a subset of the words within the vocabulary, it's necessary to aggregate multiple rows of this matrix, in order to create representation for tweets ($N \times D$ matrix).

After investigating with various ways of aggregation (concatenation, summation and averaging), mean of WEs of all the words in a tweet is found to be the most expressive one for our classification algorithms.

[2]http://github.com/maciejkula/glove-python

[3]http://nlp.stanford.edu/projects/glove/

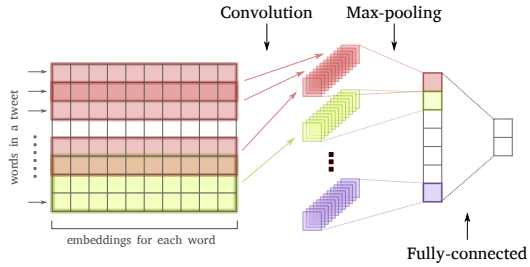


Figure 1: Convolutional Neural Networks for tweets

***Paragraph Embeddings**:* One method that constructs directly embeddings for paragraphs (tweets, in our case) is proposed by [7]. The advantage of this method is the fact that the order of the words in a tweet is also taken into account, by permuting them multiple times, which is claimed to reveal the semantic meaning of the phrases in a paragraph/tweet. There are two model representations described in this paper: Distributed bag of words (DBoW) and Distributed Memory (DM) both of which are included in our experiments. We observed that DBoW model is more efficient and requires less memory.

V. Classification

After the text data set, i.e., tweets are converted into numerical representation, remaining is the application of classification algorithms. For this purpose, we employed several algorithms such as Logistic Regression (LR), Naive Bayes (NB), Random Forest (RF), Support Vector Machines (SVMs), Neural Networks (NNs) and Convolutional Neural Networks (CNNs). After preliminary performance investigation of classification algorithms, we decided to continue our experiments with the three promising ones, which are explained below.

***Support Vector Machines**:* SVMs are very strong classification algorithms, even in high dimensional spaces, and the aim is to separate samples from different classes with a clear margin. They can be applied as both linear binary classifiers and non-linear complements after kernel trick. In our framework, we used the linear SVM classifier implementation provided by *scikit*[4], which is a well-known, open-source machine learning library in Python. After some experimentation, penalty parameter is used as 1 for 10K iterations for all training.

***Neural Networks**:* NNs, as nonlinear function approximators trained via backpropagation, are configurable structures with number of layers, number of neurons per layers and nonlinearity options. For our binary classification problem, we experimented with several network structures, some of which are presented in Section VI. Training is achieved using lbfgs algorithm, which is more memory friendly compared to SGD counterparts, with a constant learning rate of 10^{-5} for at most 1000 iteration, during which early stopping was possible when the cost function is noticed not to improve any more than 10^{-6}.

***Convolutional Neural Networks**:* As well known for its success in computer vision area in the last years, CNNs are also applied to lots of NLP tasks with success [8]. In order for CNN to be applied to text data, get trained by batches and understand the contextual relation between numerical representations of words, each tweet is assumed to be consisting of maximum number of words n_{max}. Hence, each tweet is represented by a matrix with a size of $n_{max} \times D$, where D represents the dimension of representation (*GloVe* vectors in our case). Notice that all these matrices are zero-padded in the absence of words. As we may see in Figure 1, the main idea behind is to apply a bunch of filters with different window sizes (depicted by different colors). Afterwards we convolve, decrease the number of data by *max-pooling* and apply *soft max* layer as the classifier. Although we experimented with different training strategies, the results to be presented in Section VI are trained with minibatches of size 1024, Adam optimizer with a exponentially decaying learning rate every 1000 epoch, which is initialized as 5×10^{-4}. n_{max} is chosen to be 40 for all the experiments. A dropout layer with a keep probability of 50% is used for training in order to avoid overfitting the statistics of training dataset.. Our CNN implementation relies on *Tensorflow*[5], which is developed by Google and an open source software library for numerical computation using data flow graphs.

Table I: Fully connected NN configurations

	# of hidden layers	#of neurons layer 1	#of neurons layer 2	Batch size
NN1	1	16	-	256
NN2	1	32	-	200
NN3	1	64	-	128
NN4	2	16	4	128

VI. Results

Among the exhaustive amount of tests conducted with most of the possible combinations of tweet representation and classification methodologies, we will present the results that helped us to better understand each algorithm.

First, Word embeddings were experimented with all four different options, *baseline*, *trained*, *pretrained* and *merged* as they are described in Section IV. In order to understand the performance of SVM classifier in this configuration, we also tested NN classifier for the same data representations. For this purpose, we first intended to find the best possible NN configuration for the resources we have, since more powerful architectures require more computational and memory resources. Both single hidden layer and two hidden layers architectures with different number of neurons are trained. Table I gives the architectures and corresponding abbreviations, which are used in Figure 2 to give a clear picture of performances for different GloVe embeddings and SVM-NN classifiers.

As can be seen from the performance chart, NN outperformed SVM for all cases. In addition, the best performance is obtained with the largest single layer NN architecture, i.e. NN3, which was the biggest that our resources enabled us. Second hidden layer, i.e., more nonlinearity never achieved better. Regarding the *GloVe* dictionaries, pre-trained vocabulary always lead to better results compared to trained ones, which is expected due to the amount of data used for the pre-trained vocabulary. However, merging the new words contributes to performance by 1% for both NN an SVM classifiers. Only interesting thing to notice is

[4]http://scikit-learn.org/

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

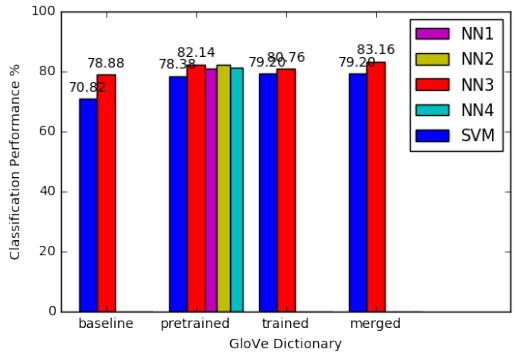
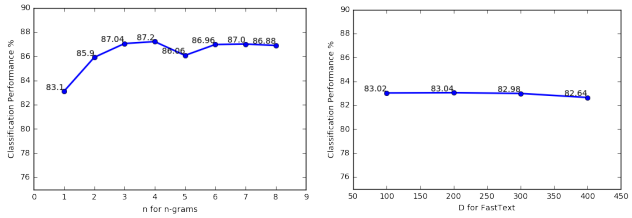


Figure 2: Performance of SVM/NN classifiers using GloVe representations obtained by a variety of dictionaries

the fact that utilization of TF-IDF constants as weighting coefficients for *GloVe* representations does not improve the classification performance. In fact, we obtained 81.50% classification accuracy with weighted representations of *pretrained* vocabulary and NN3 classifier, whereas the raw representations achieved 82.14%.

Furthermore, we investigated the performance of *n-grams* representation for different window sizes, where it is applied to preprocessed tweets and SVM is utilized as the classifier. Results are summarized in Figure 3a with 4-grams to seem to be the best representation.



(a) n-grams: Performance vs. n (b) FastText: Performance vs. D

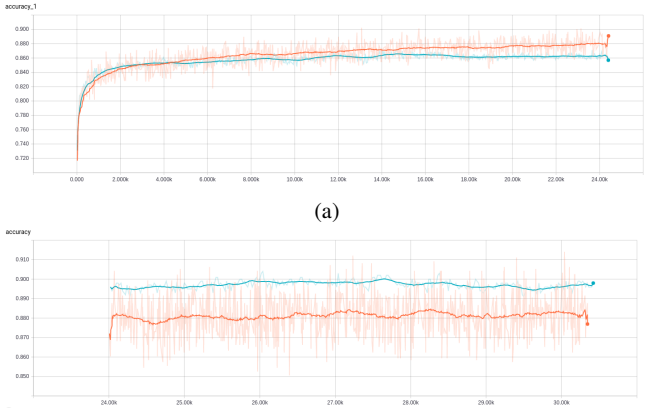
As another test, FastText algorithm is experimented for different dimensions of representations, results of which is given in Figure 3b. For all different dimensions, preprocessed dataset is utilized and the classifier is trained with a learning rate 0.05 by 50 epochs. Dimension of 200 seems to give the best accuracy.

In order to further observe the contribution of preprocessing step, we also chose the best-performing n-gram and FastText configurations to apply them to raw data. results presented in Table II clearly indicates that proposed preprocessing step adds around 2% to the classification performance.

Table II: Classification performance $(\%)$ with and without preprocessing

Algorithm	w/o preprocessing	w/ preprocessing	Improvement
4-grams	85.28	87.20	**1.92**
FastText, D=200	80.36	83.04	**2.68**

The last but not the least, we trained several CNN architectures, where we altered the capacity of the network in order see the limits of the algorithm regarding the training data. Best performance is achieved when we use a single hidden layer with 128 neurons, where 128 convolutional filters of {2,3,4,5,6} neighborhoods are used. Regarding the computational necessities, instead of cross-validation, we kept randomly chosen 10K samples of training dataset for the validation for all training attempts. Figure 4 illustrates the evolution of accuracy during the training. Roughly speaking, after 18K steps, we suspected model to overfit due to the divergence of accuracy values for training and validation dataset, but as can be seen on the second figure, when we kept training with another randomly chosen validation data due to non-constant seeds, validation performance was better than the training. The main reason behind this difference is related to the fact that second validation set had been used for the training at the first stage. Since we didn't have enough time for a training from scratch, we stopped the training at around 30K global steps, which gave us 87.54% test error, which had been 87.48% after 18K global steps.



(a)

(b)

Figure 4: Evolution of accuracy values for training(orange, smoothed) and validation(blue) datasets for 4a first 18K steps and 4b remaining 12K steps

When we increased or decreased the capacity of networks by increasing/decreasing the number of filters per neighborhood sizes as well as the number neighborhoods, or adding another hidden layer before soft-max classifier, performance always got worse. If we increased the capacity, decay in the performance is related to overfitting and if we decrease the capacity, deterioration can be due to insufficient representational power.

VII. Conclusions

In this report we described possible preprocessing steps, text representation options and test results for a variety of classification algorithms, which was followed by a concrete comparison.

As you see in Table III two of our approaches achieved top-3 scores in Kaggle's competition, with the one employing a convolutional neural network to reach the 87.54% of correct predictions.

Table III: Best Performance of Classifiers

	LR - WE (pre)	NB - TFIDF	NN - WE (pre)	FT (D=200)	SVM - TFIDF(4-grams)	CNN - WE
Classification Accuracy (%)	79.50	78.18	83.16	83.04	87.20	87.54

REFERENCES

[1] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[2] S. Khaitan, A. Das, S. Gain, and A. Sampath, "Data-driven compound splitting method for english compounds in domain names," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ser. CIKM '09. New York, NY, USA: ACM, 2009, pp. 207–214. [Online]. Available: http://doi.acm.org/10.1145/1645953.1645982

[3] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 168–177.

[4] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[5] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[6] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[7] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents." in *ICML*, vol. 14, 2014, pp. 1188–1196.

[8] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: http://arxiv.org/abs/1408.5882