Machine Learning project 2
-Tweet sentiment prediction-

Rayan Daod Nathoo,* Kopiga Rasiah,† and Yann Meier‡
(Dated: December 19, 2019)

Tweet sentiment prediction is a challenging machine learning tasks. In this project, a balanced set of 2.5 millions tweets, each of them being classified as negative or positive, is analyzed to train a machine learning model to predict the polarity of a tweet. GloVE word embedding method and Stanford's GloVe version with an extended vocabulary are used to build a tweet embedding. The best model, an neural network with 256 nodes and one hidden layer, with a specific pre-processing of the data and GloVe word embedding reaches 83.8% on a test set of 10'000 tweets and 83.4% in internal 5-folds cross validation.

I. INTRODUCTION

Twitter is a microblog type of social media where people share there opinions, sentiments and thoughts on various topics such as politics, daily life, catastrophic events, etc. with short messages of no more than 140 characters (280 characters since 2017). Analyzing the sentiment of a tweet in an automatic manner can have various applications. However, predicting the sentiment of a tweet or more precisely its polarity, negative or positive, with good accuracy is challenging and demanding machine learning task. The goal of this project is to classify tweets that initially contained positive ":)" and negative smileys ":(" which were then removed for the classification task.

In literature, predicting tweet sentiment is an active field of research and many approaches are used to transform words or sentences into numbers to run a machine learning algorithm and build a model.

The GloVe model is a popular method to build features from words. Pennington et al. (2014) [1] built this model using the frequency of occurrence of a pair of words in a sentence, text or corpus. This model allows to capture the semantic of the words. Pennington et al. also trained their model on very large data-sets to find optimal feature vectors for 1.2 million words (here referred as *Stanford's vocabulary*). This model proved to be efficient at extracting the meaning of a sentence or a text.

Alharbi and de Doncker (2019) [2] used a Convolutional Neural Network (CNN) to perform binary sentiment classification of tweets with ReLu activation function to improve both training time and accuracy. They used features such as the content of the URL cited in a tweet, the number of emoticons, the number of positive and negative words according to a lexicon as well as user behavioral information of the user to translate tweets into numbers.

The next sections address the tweet processing methods and the models used as well as the results for different models and data processing configurations. Results are eventually discussed.

TABLE I. Pre-processing steps with corresponding external Python library used.

Tasks	**external library**	**ID**
remove tweet duplicates	-	P1
correct spelling	autocorrect [3]	P2
parse hashtags	wordsegment [4]	P3
replace contractions	- [5]	P4
replace smileys	- [5]	P5
remove hooks "<...>"	-	P6
remove stopwords	NLTK [6]	P7
remove punctuation	textblob [7]	P8

II. METHODOLOGY

We are given two set of data, a balanced train set of 1.25 million positive tweets and 1.25 million of negative tweets, and a test set of 10'000 which are not classified. In both datasets, positive ":)" and negative ":(" smileys have been removed. A reduced train set of 2×100'000 is used for fast testing of new models, features, etc.

A. Pre-processing

One first necessary but complex task to perform before doing statistics is to clean the data. In general a tweet contains slang words, spelling mistakes, abbreviations, special characters, links, calls to other users (*@username*). For example:

#yougetmajorpointsif you play soccer . idc about any other sport ! just soccer (because other than basketball , that's all i understand .)

The different steps of pre-processing of the tweets are shown in table I.

B. Data statistics

Even though positive ":)" and negative ":(" have been removed, some other types of smileys remain, such as

* rayan.daodnathoo@epfl.ch
† kopiga.rasiah@epfl.ch
‡ yann.meier@epfl.ch

TABLE II. Positive and negative smileys frequency in positive and negative tweets before pre-processing.

	Positive tweets	Negative tweets
# positive smileys $\times 10^{-3}$	60.0 (69.8%)	25.9 (30.2%)
# negative smileys $\times 10^{-3}$	3.82 (35.5%)	6.95 (64.5%)

TABLE III. Vocabulary statistics for positive and negative tweets after pre-processing of the train data-set.

Positive tweets			**Negative tweets**		
Rank	word	occ. $\times 10^{-3}$	Rank	word	occ. $\times 10^{-3}$
1	i	421.6	1	i	512.6
2	you	394.8	2	the	386.8
3	to	348.7	3	to	371.8
4	the	301.3	4	a	232.5
14	love	124.8	39	love	47.4
29	good	69.3	48	miss	39.0
32	lol	63.6	59	sad	33.5
46	thanks	46.8	68	lol	28.9
50	haha	41.8	93	good	23.3
67	happy	32.0	106	hate	21.7
95	hope	22.2	114	sorry	19.2
99	great	21.4	118	bad	18.5

":P", ":d", "(:", ":-D", etc. which can give information on a tweet's sentiment.

A first approach to analyze the meaning of a tweet is to look at the vocabulary and the occurrences of words in positive and negative tweets. Table III shows the most frequent four words of each subset of tweets. The other words shown are the most frequent words which are associated with polarity and sentiment. First, most of the more frequent words are what is called "stopwords" that does not carry meaning but are essential in the structure of a sentence. Secondly, the words related to emotions, positiveness or negativeness are present in both data-set. One interesting feature is that the words "love" (which could arise also from the replacement of a positive smiley) and "good" are present in both dataset with a high frequency, even though they are more common in positive tweets. This shows that an approach only based on the polarity of single words in a tweet would not be very efficient at distinguishing tweets and evaluating the real meaning of a tweet. The first improvement that arises is to build a co-occurrence matrix which takes into account which words appears with one specific word in a specific tweet to help establish a context for each tweet.

C. Models for word embedding

1. *GloVe word embedding method*

One popular method of translating words into numbers is the Global Vectors for Word representation [1]. The GloVe model describes a word in a vector space according to its semantic. First, words of close meaning will be close in distance in the vector space whereas words which are used in completely different context will be further from one another. Second, the structure of the vector space allows for the vector from *man* and *woman* to be approximately equal to the vector from *king* to *queen* [8].

The first step consists of creating a vocabulary of size $N \times 1$ of the words present in the tweets. Only words appearing at least $t_{cut-off}$ times are taken into account to form the vocabulary. We take $t_{cut-off}$ to be five.

The co-occurrence matrix C of size N has components C_{ij} which corresponds to the number of times word i appears together with word j in the same context. The most natural definition of a context in this framework is the tweet. The matrix C is then factorized as following:

$$C \simeq W^\top Z \tag{1}$$

where W and Z are $K \times N$ matrices, K chosen such that $K << N$. Either matrix W or Z is used to generate feature vectors for each word. Specifically, the column i of matrix W has K components and is the feature vector for word i of the co-occurrence matrix. In the GloVe model, the number of occurrences x of word i happening with word j is replaced by:

$$f = \begin{cases} \left(\frac{x}{n_{\max}}\right)^\alpha & \text{if } x < n_{\max} \\ 1 & \text{otherwise} \end{cases}$$

where α and $n_{\max}$ are parameters. Pennington et al. recommends $\alpha = 3/4$ and $n_{\max} = 100$. They also trained word vectors on very large data-sets such as a data-set of 2 billion tweets to build a 1.2 million words vocabulary. As an alternative of training word vectors on our data-set, these pre-trained word vectors can be used to have a better feature representation for a given word.

Another way of transforming words into word vectors is the TF-IDF (term frequency-inverse document frequency) method [9]. The TF-IDF score of a word i in tweet j is measured as following:

$$\text{TF-IDF}_{i,j} = \frac{n_{i,j}}{length(t_j)} \cdot \log \frac{N}{d_{tf}} \tag{2}$$

where $n_{i,j}$ is the number of times word i appears in tweet j, $length(t_j)$ is the number of words contained in tweet j, N the number of tweets and d_i the number of times word i appears in the document. The input data matrix X is a N where N is the number of tweets and V is the number of word in the vocabulary. The entry X_{ji} is defined as:

$$X_{ji} = \begin{cases} \text{TF-IDF}_{i,j} & \text{if word} i \text{appears in tweet } j \\ 0 & \text{otherwise} \end{cases}$$

D. Tweet embedding

Once word embedding is performed, we are left with a collection of vectors of dimension $K \times 1$ for a given tweet and a feature vector must be attributed to the tweet. Several approaches are considered:

1. adding up every word vector of a tweet (summation) ;

2. averaging the word vector of a tweet (average).

E. Models for optimization

Figure 2 shows the distribution of the first two components of the of 500 randomly selected positive (red dots) and negative (blue dots) tweets. It is observed that the positive and negative tweets cannot be split into two groups by a separation line and even less by a straight line. Similar plots are obtained with other features (e.g. feature number 1 and 5, etc.). Although this graph only shows two features among a hundred, it gives an insight that the data may not be easily separable. A way to find an optimal non linear classification model for such a complex data-set is to use a neural network (NN). This allows to find highly non-linear "separation curves" to classify the data into one or the other group.

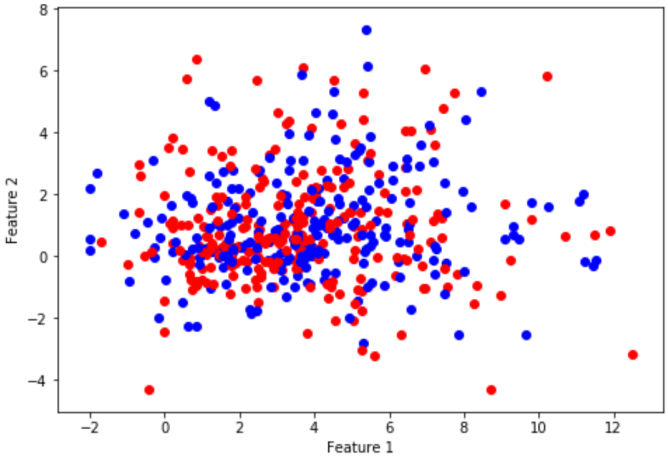

FIG. 1. First two components of tweet features for 500 tweets randomly selected within the train data-set. Word vector are calculated using GloVe and $K = 100$. Red dots indicate positive tweets whereas blue dots indicate negative tweets. Both clouds of points are superimposed and there is no clear distinction between positive and negative tweets on this 2D projection.

Table IV summarizes the word and tweet embedding methods, the optimization algorithms and cost functions.

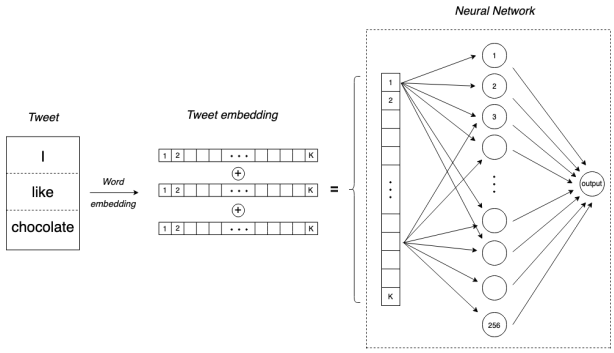

FIG. 2. Schematics of the model for the Neural Network used.

TABLE IV. Word and tweet embedding methods and optimization methods with corresponding external Python library used.

Model for word embedding	**external library**	**ID**
GloVe trained on train set	*course solution*	E1
GloVe, pre-trained word vectors	GloVe [10]	E2
TF-IDF	- [9]	E3
Model for tweet embedding		
summation	-	T1
average	-	T2
Model for optimization		
logistic regression	Scikit learn [11]	M1
NN with n hidden layers	Keras [12]	M2Ln
Cost function		
Mean-square error	Keras	C1
Mean-absolute error	Keras	C2
Logistic function	Keras	C3
Hinge	Keras	C4
Binary cross-entropy	Keras	C5

III. RESULTS

The results of the different models and the sensitivity analysis are shown in table V. The 5-Folds cross-validation (CV) is performed on the small dataset. The sensitivity analysis is performed by varying one parameter at a time. The first simple model implemented is logistic regression with mean-square error and $K = 150$. The accuracy obtained in the CV is 61.4% and is taken as a baseline. The second step consisted of using Pennington's word vectors [1] which significantly increased the accuracy. Then, K is varied. The next results are obtained with a one-layer neural network with 256 nodes. The activation function used within the neural network is always Relu and the sigmoid function is used for the last node in the output layer. Regarding the optimizer, Adams from the Keras library. The batch-size used by the optimizer is Adams.

The best model is obtained with mean-square error cost function with 20 epochs. The accuracy reaches **83.6%**in the train set and **83.8%** in test set when prediction are submitted on Alcrowd.com.

TABLE V. Table of results for the different models and their parameters. Accuracy in parenthesis indicates that the cross-validation (CV) is performed on the full dataset.

Model		5-Folds CV accuracy
P1-6 + E1 + T2 + M1 + C1, K = 150 (Baseline)		0.614 (0.587)
P1-6 + E2 + T2 + M1 + C1		
	K = 25	0.695
	K = 50	0.725
	K = 100	0.759
	K = 200	0.779
P1-6 + E2 + T2 + M2L1, K = 100, 10 epochs		
	C5	0.769
	C1	0.746
	C2	0.697
	C4	0.718
P1-6 + E2 + T2 + M2L1, K = 200, 10 epochs		
	C5	0.784
	C1	0.786
	C2	0.760
	C4	0.758
P1-6 + E2 + T2 + M2L1, K = 200		
C5	10 epochs	0.784
	20 epochs	0.804 (0.834)
	30 epochs	0.787
C1	10 epochs	0.786
	20 epochs	0.815 (0.836)
	30 epochs	0.816

IV. DISCUSSION AND CONCLUDING REMARKS

We started as baseline with a logistic regression on preprocessed data, using GloVe which optimizer is a gradient descent. With this baseline, the accuracy obtained using 5-fold cross-validation was 61.35%. As stated before, this lack of accuracy can be explained due to the complexity of the data distribution or a poor construction of vector embedding. We looked for pre-trained word vector and chose the GloVe from Pennington with different dimensions K, where the best accuracy was achieved at 0.779 for K=200 and 0.769 for K=100. We therefore chose to keep the Stanford GloVe word vectors. Because of the complexity of the data distribution, we decided to switch to neural networks, rather than a regression. With the help of Keras library, different cost function have been tested for the neural network (cf. Table V), and the best accuracy reached 0.786 with Mean-square error for K=200 and 0.769 with binary cross-entropy for K=100. We first defined the epochs to 10 to run quick simulations. After several tests on parameters, we decided to increase the number of epochs, since theoretically increasing the number of epoch should increase the accuracy on the train set (without over-fitting !). An early stopping function [12] stops the iteration as soon as the CV accuracy from the test subset and the train subset of the train set diverges, i.e. when the test error starts to increase (overfitting).

Further improvements could be to integrate a working CNN. This particular type of neural networks is well suited for a text analysis as it applies a filter which considers only a certain number of words away of the word of interest to build a context. As a tweet may convey complex meaning and natural human language may includes contradictions or not clearly defined opinion, a CNN is seen as an efficient model to use in a sentiment analysis and could further improve the performance. Beside this, preprocessing must also be considered. Stop words were not removed (P7) as they seem to still participate to the semantic of a tweet. For example, the word "not" is a stop word but removing it from a sentence where it appears in front of "good" would completely change the meaning of that sentence. This idea that stop words may not be so negligible is confirmed by the fact that the accuracy is lower

Another idea to take advantage of the smiley list would be to build a new feature column which counts the number of times positive and negative smileys are found in a tweet.

[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.

[2] A. S. M. Alharbi and E. de Doncker, Twitter sentiment analysis with a deep neural network: An enhanced approach using user behavioral information, Cognitive Systems Research **54**, 50 (2019).

[3] J. McCallum, Autocorrect 0.4.4 Python library, `https://pypi.org/project/autocorrect/` (2019).

[4] G. Jenks, Wordsegment 1.3.1 Python library, `https://pypi.org/project/wordsegment/` (2018).

[5] charlesmalafosse/FastText-sentiment-analysis-for-tweets.

[6] E. K. Bird Steven, Edward Loper, Natural language processing with python, (2009).

[7] S. Loria, Textblob 0.15.3 Python library, `https://pypi.org/project/textblob/` (2019).

[8] See [10] for a brief description of what GloVe does.

[9] C. Maklin, TF IDF | TFIDF Python Example (2019).

[10] GloVe: Global Vectors for Word Representation.

[11] scikit-learn: machine learning in Python — scikit-learn 0.22 documentation, `https://scikit-learn.org/stable/` (2019).

[12] Home - Keras Documentation, The Python Deep Learning library, `https://keras.io/` (2019).