Twitter Sentiment Analysis

Ioannis Ioannidis
University of Piraeus
NCSR Demokritos

Abstract: This project addresses the problem of sentiment analysis, that is classifying text according to the sentiment expressed in them: positive or negative. Texts are drawed from Twitter, an online micro-blogging and social-networking platform which allows users to write short status updates of maximum length 140 characters. In that way, we will train several classifiers and keep the one with the best performance. This classifier will be used in order to build a simple web-app, which will be returning live feedback for a specific domain, using this topic's hashtags to extract the latest tweets on it and predict the sentiment.

1. Introduction

Sentiment analysis, also refers as opinion mining, is a sub Machine Learning task where we want to determine which is the general sentiment of a given document. Using Machine/ Deep Learning techniques and Natural Language Processing we can extract the subjective information of a document and try to classify it according to its polarity such as positive or negative. It is a really useful analysis since we could possibly determine the overall opinion about many domains such as selling objects, or predict stock markets for a given company like, if most people think positive about it, possibly its stock markets will increase, and so on. Sentiment analysis is actually far from to be solved since the language is very complex (objectivity/subjectivity, negation, vocabulary, grammar,...). However this is the main reason it constitutes such an interesting domain to work on.

In the present report we describe how we tried to compare Machine Leaning(ML), Deep Learning (DL) and pretrained models to determine which performs better on our data. The structure of the document is as following: Firstly, in the *Data Description* we provide some information about the data we used for the training and validation procedures, while in the *Theoritical Background* section we give an intuitive taste of the algorithms we will be using and comparing. *Experiments* part describes the experimental setup and provides the final comparison results. Closing, all the remarks highlighted during the task and the observations we ended up with are quoted in the *Conclusion*.

2. Data Description

To gather the data many options are available. Some researchers, in previous works, built a program to collect automatically a corpus of tweets based on two classes, "positive" and "negative", by querying Twitter with two type of emoticons:

- Happy emoticons, such as ":)", ":P", ":)" etc.
- Sad emoticons, such as ":(", ":'(", "=(".

Others made their own dataset of tweets by collecting and annotating them manually which very long and fastidious.

In our case, we found an annotated twitter corpus on Kaggle[1]. The corpus was quite large and unbalanced. In order to restore the balance in labels, as well as keep the training time in reasonable levels we kept a total of 40000 (20000 positive and 20000 negative) tweets for training.

[1] https://www.kaggle.com

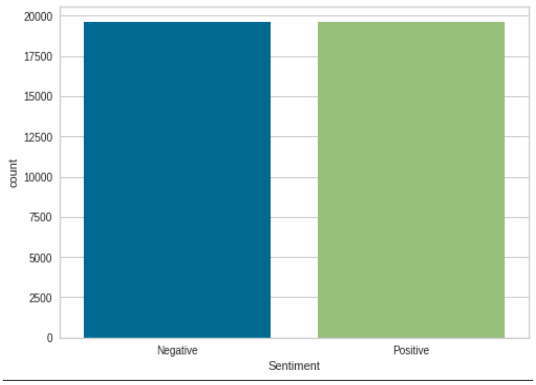

Fig. 1: Positive and Negative Label Barplot

Before passing tweets to each model, we applied a clean-up function, which turns all letters to lowercase, removes punctuation, stopwords, links, emojis and stems each word, in order to keep only the high level information and get better results.

Final dataset has the following form:

	Tweet	Sentiment
0	read complic superfici unconclus distract scat...	Negative
1	show desper congress comeback power parti fund...	Negative
2	sir may good pick begin disappoint toward way ...	Negative
3	would think latter given campaign ad welfar re...	Negative
4	mayb worri modi bomb belov pakistan kill hafee...	Positive
...	...	...
39995	congress releas list star campaign first secon...	Positive
39996	know want someth fake defam nehru modi defend ...	Negative
39997	shame guy like shesan past today work elect ch...	Positive
39998	sure stop work abl give current earn spend ful...	Positive
39999	live poll what public opinion present narendra...	Positive

39174 rows × 2 columns

Fig. 2: Final Dataset

3. Theoretical Background

3.1. Machine Learning Algorithms

3.1.1. Naive Bayes

Naïve Bayes classifier is a machine learning model that applies the Bayes theorem, presented in Eq. (1), for probabilistic classification. By observing the values (input data) of a given set of features or parameters, represented as B in the equation, Naïve Bayes classifier is able to calculate the probability of the input data belonging to a certain class, represented as A.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

For the classification of input data to take place, the probabilities of it belonging to each of the existing classes must be determined and the one with the highest probability will be the class to which the input

data belongs. Therefore, class a with the highest probability must be found as expressed in Eq. (2), where b_i is one of the n features/predictors observed.

$$a = argmax_a P(a|b_1, ..., b_n) \tag{2}$$

Naive Bayes is frequently used in natural language processing (NLP) problems. It predicts the tag of a text by calculating the probability of each tag for a given text and then output the one with the highest percentage.

3.1.2. K - Nearest Neighbors

K-Nearest Neighbors (KNN) is a standard Machine Learning method that has been extended to many classification tasks, including sentiment analysis. The idea is that one uses a large amount of training data, where each data point is characterized by a set of variables. Conceptually, each point is plotted in a high-dimensional space, where each axis in the space corresponds to an individual variable. When we have a new (test) data point, we want to find out the K nearest neighbors that are closest (ie, most "similar" to it) as shown in Fig.3.

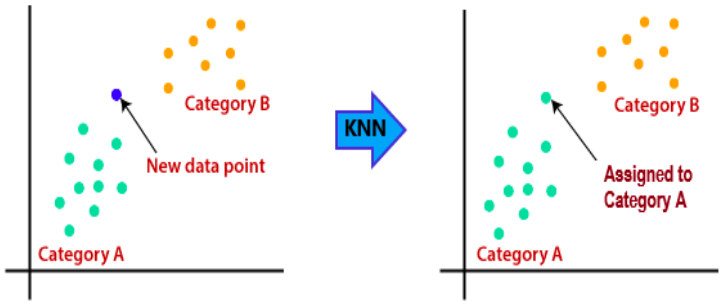

Fig. 3: KNN Algorithm 2-D Representation

3.1.3. Support Vector Machines

Support vector machine (SVM) is a method for the classification of both linear and nonlinear data. If the data is linearly separable, the SVM searches for the linear optimal separating hyperplane (the linear kernel), which is a decision boundary that separates data of one class from another. Mathematically, a separating hyperplane can be written as $W \cdot X + b = 0$, where W is a weight vector and $W = w_1, w_2, ..., w_n$, X is a training tuple and b is a scalar. In order to optimize the hyperplane, the problem essentially transforms to the minimization of $|W|$, which is eventually computed as $\sum_{i=1}^{n} a_i y_i x_i$, where a_i are numeric parameters, and y_i are labels based on support vectors X_i. That is: if $y_i = 1$ then $\sum_{i=1}^{n} w_i x_i >= 1$; $if y_i = -1$ then $\sum_{i=1}^{n} w_i x_i >= -1$.

If the data is linearly inseparable, the SVM uses nonlinear mapping to transform the data into a higher dimension. It then solve the problem by finding a linear hyperplane. Functions to perform such transformations are called kernel functions. The kernel function selected for our experiment is the Gaussian Radial Basis Function (RBF): $K(X_i, X_j) = e^{-\gamma ||X_i - X_j||^2/2}$, where X_i are support vectors, X_j are testing tuples, and γ is a free parameter that uses the default value from scikit-learn in our experiment. Figure 9 shows a classification example of SVM based on the linear kernel and the RBF kernel.

3.2. Deep Learning Model - LSTM

LSTM can be regarded as a modified Recurrent Neural Network (RNN). The memory of past input is important in the sequence learning problem. LSTM was specially designed and modified to solve the vanishing gradient and explosion gradient problem in long-term training. Due to the memory cell, the

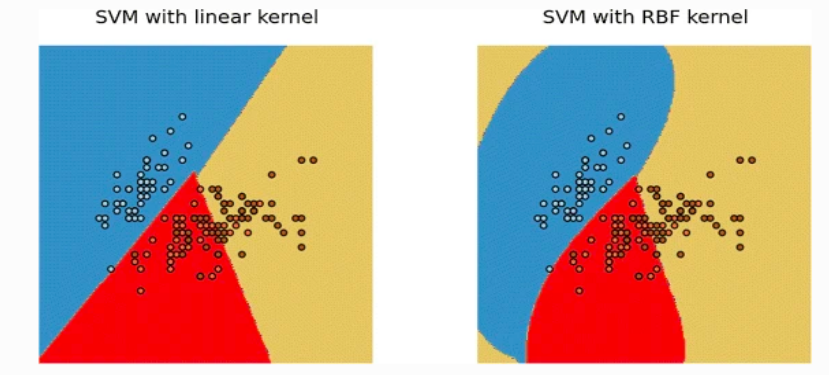

Fig. 4: Classification Example of SVM

LSTM can maintain the error values and continue the gradient flow. Therefore, the vanishing problem is eliminated and information can be learned from hundreds-time steps long sequences. LSTM has better performance than other RNN architectures because the vanishing gradient problem is alleviated.

The main difference between the LSTM and RNN is that a series of memory gates are added, including the forget gate, input gate, and output gate. In the training process, some less important information is thrown away for the memory cell to make room for information that is newer and more relevant. The forget gate is designed to solve this problem. The forget gate deletes or maintains the information by multiplying the value in the memory cell by a number of 0 or 1. If a value needs to be preserved for many steps in the memory cell, the input gate or write gate is added to the LSTM structure. And the output gate is added to solve the problem that multiple memories are against each other. After knowing the memory gates, the LSTM neural network can be expressed.

The input data are set as x^t for the time t. The number of memory cells is set as N in LSTM. The dimension of the input data is set as M. The feedforward process of the LSTM algorithm can be explained as follows:

$$i^t = g(w_i x^t + p_i y^{t-1} q_i \cdot c^{t-1} + b_i) \tag{3}$$

$$l^t = \sigma(w_l x^t + p_i y^{t-1} q_i \cdot c^{t-1} + b_l) \tag{4}$$

$$f^t = \sigma(w_f x^t + p_i y^{t-1} q_i \cdot c^{t-1} + b_f) \tag{5}$$

where:

- i^t is the activation of the input gate, l^t is the activation of the output gate and f^t is the activation of the forget gate.
- $w_i, w_l, w_f \in R^{N \times M}$ are the input weightings of the LSTM.
- $p_i, p_l, p_f \in R^{N \times M}$ are the input weightings of the LSTM.
- $q_i, q_l, q_f \in R^{N \times M}$ are the weightings of the memory cells.
- $b_i, b_l, b_f \in R^{N \times M}$ are the biases of the corresponding activations.

After that, the value of the output gate and the state of the memory cell can be calculated as follows:

$$c^t = i^t \cdot l^t + c^{t-1} \cdot f^t \tag{6}$$

$$o^t = \sigma(w_o x^t + p_0 y^{t-1} + q_0 \cdot c^t + b_0) \tag{7}$$

where c^t is the state of the memory cell and o^t is the value of the output gate and w_0, p_0, q_0, b_0 are the weightings and bias of the output gate. Finally, the output value of the cell can be calculated as follows:

$$y^t = h(c^t) \cdot o^t \tag{8}$$

Among the above equations, $g(x), \sigma(x), h(x)$ are the activation functions of the input gate, forget gate and output gate respectively. In general, the sigmoid function and tanh function are set as the activation functions, which are given as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{9}$$

$$g(x) = h(x) = tanh(x) \tag{10}$$

3.3. BERT Model

BERT is an NLP preprocessing model based on neural network. Compared with the traditional language model word2vec, BERT is more flexible and more advantageous and can be used as a substitute for word2vec. word2vec represents each word in a fixed form and does not change with context. That is, the representation has nothing to do with the context in which the word appears, but due to the complex semantic characteristics of natural language itself, the meaning of the same word in different contexts may be different, so using word2vec for word vector representation will reduce the accuracy of the results of downstream tasks. The BERT can dynamically adjust the word vector representation according to the context in which the word is located. Since the birth of BERT, it has set new records in 11 natural language processing tasks, such as MultiNLI, QQP, and so forth. Besides, the word vector obtained by using BERT has higher quality features. Inputting these word vectors with higher quality features can achieve better results in downstream tasks.

Due to the complex structure of BERT, a lot of training time and expensive training costs are required. However, Google has opened its source code. Users only need to finetune the pre-trained BERT model according to the actual task to use, which greatly saves training time and cost. The trained BERT model can be applied to a variety of different natural language processing downstream tasks and only needs to be fine-tuned in various degrees. In the sentiment analysis task, a [CLS] symbol is added to the pre-trained BERT model, and the symbol is inserted at the forefront of the input text. The output result of the symbol can be regarded as the semantic representation of the entire input text, that is, use the output of the symbol as the result of sentiment classification. Because the symbol is different from other words in the text, this symbol independent of the input text does not carry obvious semantic information, so it will output the result more objectively and impartially. The output can effectively integrate the semantic information carried by each word in the input text and then better represent the text. The model structure of BERT is shown in Fig.4.

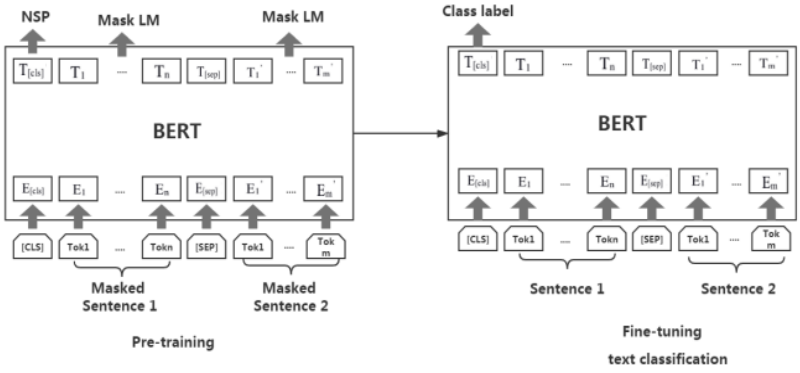

Fig. 5: BERT Structure

4. Experiments

4.1. Experimental Setup

In order to evaluate each model's performance we will be focusing on the metrics resulting from the confusion matrix. Specifically, we will measure precision, recall and F1-score and we will keep the one

with the best outcome.

Precision is the number of True Positives (TP) (the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class.

$$Precision = \frac{TP}{TP+FP}$$

On the other hand recall is the number of True Positives (TP) divided by the total number of elements that actually belong to the positive class.

$$Recall = \frac{TP}{TP+FN}$$

A precision score of 1.0 means that every result retrieved was relevant (but says nothing about whether all relevant elements were retrieved), whereas a recall score of 1.0 means that all relevant documents were retrieved (but says nothing about how many irrelevant documents were also retrieved).

There is a trade-off between precision and recall, where increasing one decreases the other and we usually use measures that combine precision and recall, such as F1-score. F1-score can be interpreted as the weighted average of the precision and recall and its a good measure of a model's accuracy.

$$F1 = \frac{2 \times (precision \times recall)}{precision + recall}$$

4.2. Results

For the evaluation procedure, we kept aside a set of 35000 samples. Those tweets are passed through each algorithm and here we report the achieved performance on the metrics presented in *Experimental Setup*.

	Accuracy	Precision	Recall	F1-Score
Naive Bayes	68.53%	0.73	0.59	0.65
KNN	62.92%	0.6	0.79	0.68
SVM	80.33%	0.83	0.76	0.79
LSTM	76.15%	0.81	0.68	0.74
BERT	84.52%	0.85	0.83	0.84

Table 1: Results

As we expected the pretrained model BERT performs better in all categories and achieves remarkable overall scores.

Note that in code notebooks more plots and details for each algorithm are being presented.

5. Conclusion

Nowadays, sentiment analysis or opinion mining is a hot topic in Machine Learning. We are still far to detect the sentiments of a corpus of texts very accurately because of the complexity in the English language and even more if we consider other languages such as Chinese.

In this project we tried to compare the performance of five model's. All results were quite good, considering the large amount of validation data. We could further improve our classifier by trying to extract more features from the tweets, trying different kinds of features, tuning the hyperparameters, or combine models. All those procedures are left for future development.