UNIVERSIDAD DE CANTABRIA

Natural Language Processing

G687

Twitter Sentiment Analysis

Víctor López Blanco
&
Carlos Velázquez Fernández

May 27, 2023

1 Introduction

Nowadays, a huge excess of information is present all around the globe. Informatization is taking place in every sector of our society, from health and public administrations to industry and tourism. As a consequence, the need of processing and analyzing all these data appears imperiously.

We will focus in a particular case of this problem: classification of social media posts. We will be studying a binary classification (positive/negative) of tweets from the Twitter platform. This will allow us to determine whether a short text reflects good or bad emotions, which can be useful for product reviews, summarizing feedback from a community, diagnosing mental diseases or many more topics.

The main idea of the project revolves around **Sentiment Analysis**, which is the use of natural language processing to extract information, such as emotions or attitudes. In the following sections, we will explain how to pre-process the data and discuss about some of the main algorithms used to accomplish this analysis.

2 Dataset

In order to perform the sentiment analysis, we need a great amount of labelled tweets so the classifiers could train the models correctly. For this purpose, we are using Sentiment140 dataset [1]. Although its web page is not currently working properly, data can still be accessed from Kaggle [2].

This dataset contains 1.600.000 tweets extracted using the Twitter API. Each entry gives information of many features of the tweet, e.g., the user, date or identifier. However, we only need two fields:

- *Text*: The tweet itself (non-processed). The content is always in English and with no emoticons.
- *Target*: The polarity of the tweet. It can be 0 (if it is classified as negative) or 4 (if it is positive).

3 Tweets preprocessing

Tokenizing the sentences is not an easy task due to all the parameters we must take into account. We don't only find words in texts, but telephone numbers, URLs or negations too. In order to accurately classify the information, those elements must be correctly handled, so a preprocessing phase is needed.

In order to perform that task, we have built a simple tokenizer, which performs all the tasks necessary to tokenize, normalize and prepare the data for training

3.1 Token distinction

When separating the words into tokens, we have followed a series of rules to determine how to do it in a correct way. The set of rules in their regular expression form consists on:

```
# Phone numbers:
r"""(?:(?:\+?[01][\-\s.]*)?(?:[\(]?\d{3}[\-\s.\)]*)?\d{3}
[\-\s.]*\d{4})"""

# HTML tags:
 r"""<[^>]+>"""

# Twitter username:
r"""(?:@[\w_]+)"""

# Twitter hashtags:
r"""(?:\#+[\w_]+[\w\'_\-]*[\w_]+)"""

# URL
r"""http[s]?://(?:[a-zA-Z]|[0-9]|[£-_@.&+]|[!*\(\),]|
(?:%[0-9a-fA-F][0-9a-fA-F]))+"""

# Remaining word types:
# Words with apostrophes or dashes.
# Numbers, including fractions, decimals.
# Words without apostrophes or dashes.
# Ellipsis dots.
# Everything else that isn't whitespace.
r"""(?:[a-z][a-z'\-_]+[a-z])|(?:[+\-]?\d+[,/.:-]\d+[+\-]?)|
(?:[\w_]+)|(?:\.(?:\s*\.){1,})|(?:\S)"""
```

Most of these regular expressions are extracted from Christopher Potts sentiment tokenizer [3] and allow us to find all the elements above in the text and set them as individual tokens.

3.2 Lowercase

Similar words must be considered as the same regardless if the are in uppercase or lowercase. In this case, we have decided to lowercase all the tokens.

We can easily do that just by invoking the function *lower*(), present in any string.

3.3 Punctuation signs normalization

Repeating multiple times a punctuation sign is something typically used to emphasize a sentence. We can find texts as “Hello!!!!” or “What???” in our data set that must be normalized. To achieve that, we have substituted each match of two or more punctuation signs by a word representing its meaning (”exclamations”, ”questionMarks”, etc.).

In order to accomplish this, we must use more regular expressions, which will search for any repetition of two or more punctuation signs, except in the case of the periods, which we’ll search for three or more, to differentiate between ellipsis and regular stops.

3.4 Substitution

We can distinguish two main cases where the token doesn't give us as much information as if we translated it into a more general form: *users* and *URLs*.

We don't need to know which particular web pages or users are mentioned in a context, but the fact that they are mentioned by itself. This is why we have changed every match of these tokens by labels named "user" or "url".

Furthermore, there is another case, the *topics*. In twitter, every topic begins with a hashtag and it is a feature that gives us relevant information, so we only remove the symbol "#" and preserve the remaining part.

To accomplish this, we take all the text, divide it in tokens, and start processing them one by one to check their starting characters, so we can apply the substitutions required.

3.5 Word normalization

In the colloquial language we can find words with repeated letters that should be inspected. For example, the tokens "perrrrfeeeect" and "perfect" should be considered as the same.

To perform this normalization, in the same loop that we have for substitution, we have built a basic algorithm that removes a letter (if it appears multiple times) in each iteration and checks if the new word is in the dictionary. If it is known word, it returns that word. Else, it tries to remove another letter. If it finishes without reaching a known word, it returns the original token.

3.6 Negation

Negation changes completely the meaning of the sentence to the opposite, so must be taken into account. Luckily, this phenomena can be handled in many different ways. We have chosen the option of adding the prefix NOT_ before every word from the negation expression until the next punctuation sign. By this way, a sentence as "I don't like coffee" changes to "I don't NOT_like NOT_coffee".

Again, we have to iterate over all tokens, and if we encounter any negation, we start adding the prefix "NOT_" to every token until we reach a punctuation mark. Any token that is not affected by the negation remains the same way.

After following this set of steps, the data is ready to be used in training and classification.

4 Classification models

In order to perform the training, first we must split out our dataset in training data and test data. We have not taken into consideration using a third split for cross validation. Our training partition will be 80% of the total of our dataset, and the remaining 20%, will be our test data. For testing and data collecting purposes, we

will establish a parameter p, which will be the percentage of tweets that we will use. Given that they are ordered according to their target, we have decided picking the tweets randomly. A random number is generated and if it is bigger than the parameter p selected, we will skip that tweet in the reading process. By this way, we ensure that a similar amount of tweets is read from both categories, as well as we can check that the algorithms work well regardless of the particular tweets in the training and testing sets.

All the models that we will be using need the count of all the words, and it is something we do not currently have. In order to count the words, we will be using a library from Sklearn called CountVectorizer, which will count all the instances of a word and represent them in a big matrix, which will be used later. It will be configured to perform a binary count, where only will take into consideration the first instance of a word, and ignore the others. Also, it has integrated the function of ignoring the stop words of the English vocabulary, because words like "you" or "and" don't give much relevant information. We have to be cautious about which dictionary of stop words we are using, because depending on the context, we may be skipping relevant words that the author of that dictionary considers irrelevant for his objectives.

With all of that out of the way, we can use now different models to see which one performs the best in the task of predicting if a specific tweet (taken from the test partition) is negative or positive.

4.1 Naïve Bayes

Naïve Bayes algorithm is a popular machine learning technique due to its simplicity and good performance. In this case, we will be using the multinomial Naïve Bayes variant, which represents each tweet as a vector of word counts (each position contains the count of occurrences of the word in the text) [4]. Here, the probability of a vector will be given by a multinomial distribution [5].

In our example, we can consider two classes: positive and negative. First we must calculate the probability of each class $c_j \in C$:

$$P(c_j) = \frac{Number\ of\ tweets\ in\ class\ c_j}{Total\ number\ of\ tweets}$$

After computing these numbers, we will calculate the probability of each word of each tweet belonging to a class in order to estimate the resulting class. For a better comprehension, the following steps and definitions should be followed:

- Set α to the desired smoothing value.
- For each word type w in $tweet_j$, retain only a single instance of w.
- $Text_j$ = single doc containing all $tweets_j$
- For each word w_k in vocabulary:
  - n_k = number of occurrences of w_k in $Text_j$
  - n = number of tokens in $Text_j$

– $P(w_k \mid c_j) = \frac{n_k + \alpha}{n + \alpha \mid Vocabulary \mid}$

Once all the above probabilities have been calculated, we will compute the class of each new tweet removing its duplicates words and using the equation:

$$c = argmax_{c_j \in C}\ P(c_j)\ \prod_{k \in positions}\ P(w_k \mid c_j)$$

4.2 Decision tree learning

Decision tree learning [6] is a technique for classification where we just create a classification tree, with leaves presenting each class. In our case, the classes will be positive or negative.

A decision tree can be built by making subsets of our original test data, where we split in partitions depending on functions (with tweets the functions will be in relation with the counts of the different words of the test data, e.g: is $count(\text{“}sad\text{”}) > 2$). This tree is made by recursion, and will stop when partitioning the subsets doesn't add any more value to the predictions.

4.3 Logistic regression

Logistic regression is a model that takes a vector of variables and evaluates coefficients or weights for each input and then predicts the class (positive/negative) of the tweet as a word vector [7]. This model is created using the sigmoid function:

$$g(z) = \frac{1}{1 + e^{-x}}$$

We will establish the classification threshold for $h_\theta(x)$ to 0.5, and if $h_\theta(x)$ is greater that 0.5, the tweet will be positive, and otherwise the tweet will be negative. $h_\theta(x)$ is calculated as follows:

$$h_\theta(x) = g(\theta_0 + \theta_1 * x + \theta_2 * x_2 + ... + \theta_k * x_k$$

Where θ_i are the constants that the model learns from the training data, x_i is the count of the word i, and there are k words at best.

4.4 Support Vector Machines

Support Vector Machines (SVM) [8] consist on the idea of being able to distinguish classes by creating an hyperplane where the data can be separated by a line, and then classified depending whether the element is in one side or the other.

We will be using the linear kernel, as it is fitted for bigger datasets than the ones with other kernels. We will represent each tweet as the count of its words as points x in this hyperplane, which will be represented as:

$$w^T * x - b = 0$$

where w is the normal vector of the plane.

We will need to find the line where the distance the margin is maximized, being that the region that separates the line from the points.

5 Results

In order to check which algorithm from the previous shown fits better for this particular application, we have calculated their accuracies and their computation times. As we are looking for a result that works reasonably well for both, big and small data sets, the number of tweets used for training and classifying the model has been taken into account for the comparisons.

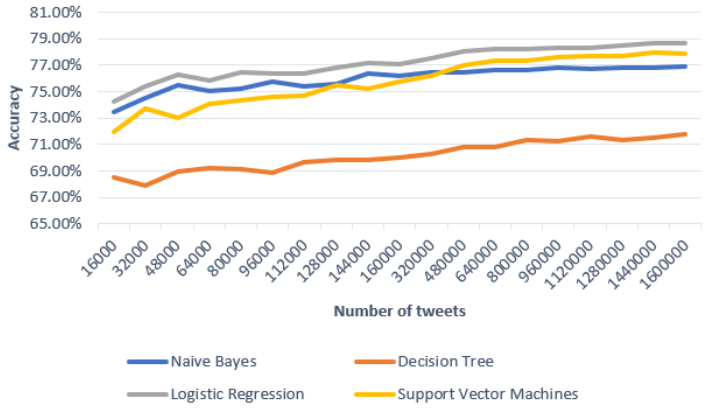

Figure 1: Accuracy of classification models.

In this plot, we can see the accuracy of the different models varying the size of the data. Firstly, we execute it with sizes in range [16,000, 160,000] with an increment of 16,000. After that, the range will be [160,000, 1,600,000] with an increment of 160,000. In other words, we compute at the beginning from 0 to the 10% of the data with a step of 0.01, and later from 10% to 100 with a step of 10%.

When talking about accuracy of the predictions, all models obtain similar results, except for Decision Tree, which is slightly worse. All of them are situated between the 68% and 78% of precision. As we will see now, the big difference lies in the time.

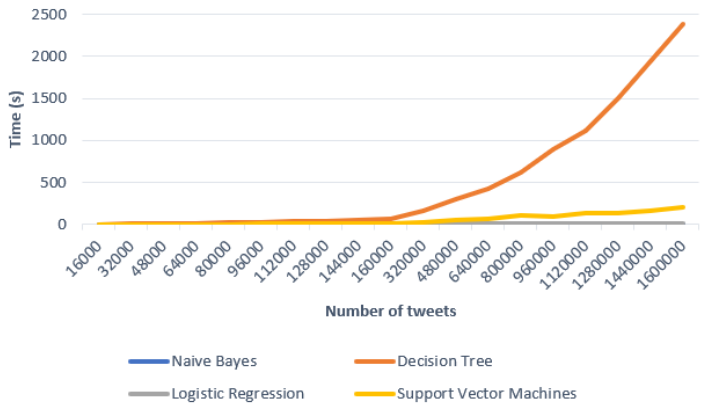

Figure 2: Increment of time over number of tweets.

Here is easy to say which algorithms are worthy to use for this case. Decision Tree does not scale well to large amounts of data. However, Naïve Bayes, Logistic Regression and SVM work perfectly fine and accomplish their purpose in a efficient

way (even though SVM can be slower than the others).

The final results with the complete dataset are:

Algorthim	**Accuracy (%)**	**Time (s)**
Naïve Bayes	76.95	2.81
Logistic Regression	78.64	14.70
Decision Tree	71.76	2379.01
SVM	77.89	204.73

Table 1: Data comparison using the whole dataset.

According to the results of the experiment, we can conclude that the best models for performing a sentiment analysis over Twitter posts are Naïve Bayes (being this one the fastest) and Logistic Regression (being the most accurate).

References

[1] Go, A., Bhayani, R. Huang, L. *Sentiment140*, 2013, http://www.sentiment140.com/.

[2] Michaelidis, M. *Kaggle*, 2017, https://www.kaggle.com/datasets/kazanova/sentiment140.

[3] Potts, C. *Stanford Linguistics*, 2011, http://sentiment.christopherpotts.net/code-data/happyfuntokenizing.py.

[4] Schneider, KM. (2004). "On Word Frequency Information and Negative Evidence in Naive Bayes Text Classification". In: Vicedo, J.L., Martínez-Barco, P., Muńoz, R., Saiz Noeda, M. (eds) Advances in Natural Language Processing. EsTAL 2004. *Lecture Notes in Computer Science()*, vol 3230. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30228-5_42.

[5] Tîrnăucă, C. (2023). *Natural Language Processing: Sentiment Analysis* [PowerPoint slides]. Faculty of Science, University of Cantabria.

[6] "Decision tree learning", Wikimedia Foundation, 17 May 2023, https://en.wikipedia.org/wiki/Decision_tree_learning

[7] A. Prabhat and V. Khullar, "Sentiment classification on big data using Naïve bayes and logistic regression," 2017 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2017, pp. 1-5, doi: 10.1109/ICCCI.2017.8117734.

[8] "Support vector machine", Wikimedia Foundation, 22 May 2023, https://en.wikipedia.org/wiki/Support_vector_machine