Machine Learning Project 2 : tweets classification

Cécile Chavane, Jehan de Bryas, Antoine Goupil de Bouillé
Projects EPFL, Machine Learning Course, Fall 2022
https://github.com/TheGreatJanus/ML-project2-text-classification.git

***Abstract*—This report develops the preprocessing and modeling to binary classify Tweets based on the feeling their convey. This report will explore the different methodologies and results that we achieved.**

I. Introduction

This second project aims at making supervised binary classification of tweets according to the emotion given by this phrase. These tweets initially contained positive smiley ":)" or negative smiley ":(". These emojis has been removed from the tweets and the tweets has been labeled according to the emoji removed.

In order to classify this text data, we need to develop a methodology to transform these tweets into matrices, then build classifiers trained on the data, to finally be able to make predictions.

Our understanding of the transversal objectives of the project seems to be structured around several key points: understanding the functioning of the basic machine learning mechanisms: preprocessing of data, visualizations, variables, selection of the best drivers, elaboration of models, selection and evaluation of models and parameters, training on the whole data set and finally predictions on the test data.

We believe that this project has many real world applications. Social medias aim to reduce anger, racism or other forms of discrimination, and therefore use algorithms to detect such form of speech. This kind of tool can also allow to detect fake news information for example.

Our report will try to clarify the functioning of our classification model, by evoking particularly the decisions of preprocessing and methods of selection of the models, allowing to reach the results that we had.

II. Vizualisation of the data

A. Vizualisation of vocabulary between positive and negative tweets

Fig. 1. represents the most common words in the positive Tweets whereas Fig. 2. gives the most common words in negative Tweets.

B. Vizualisation of the length of tweets (words number)

Fig. 3. is a histogram of the length of tweets, according to their labels.

Fig. 1. Most common words in positive tweets

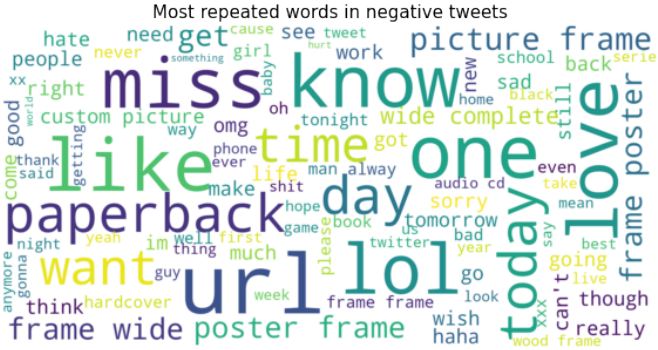

Fig. 2. Most common words in negative Tweets

III. Preprocessing the dataset

Preprocessing of the dataset is mandatory in Natural Language Processing (NLP) in order for the machine to be able to understand text. The main goal is to apply a mapping between words in the Dictionnary (the list of all words in the training dataset) and numbers, to be able to transform sentences into Matrices, in the vectorial space of our Dictionnary.

Other transformation of the data can be very useful to improve our machine learning predictions. It is mandatory to apply the same transformations to the whole data set (training and testing data).

In this part of the report, we will explain the transformation that we used and the results of each of theses preprocessing methods.

A. Cleaning the Data Set

The process of cleaning the data is mandatory and will be common to all models and preprocessing methodologies.

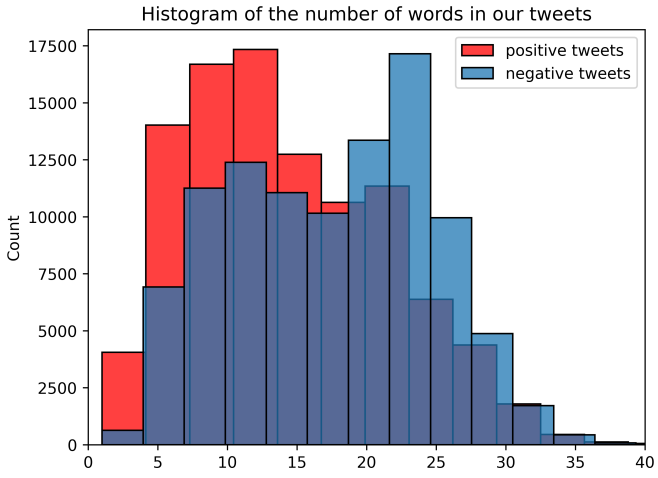

Fig. 3. Histogram of the number of words of labeled tweets

1) Labelling the data: When the Tweet corresponds to a positive emotion, we assign the label 1. When the Tweet corresponds to a negative emotion, we assign the label -1.

2) Text cleaning: We have had to clean the text data in Tweets in order to reduce the number of characters non-usefull for the understand on the sentences. Therefore, we applied text modifications to delete all special characters (e.g. "/", "?", "!", etc.), transform all words to low characters, delete single letter words. We also delete url links and numbers, because they do not convey usefull information for sentimental analysis.

3) Removing duplicates: Moreover, we have chosen to delete the duplicate Tweets (most of them coming from Re-Tweets of famous people's Tweets). Indeed, theses repeated values would give extra important to some words compared to others and therefore reduce the prediction ability.

B. StopWords

Stopwords are the words in a stop list which are filtered out during preprocessing. There is no single universal list of stop words, and the choice of the list depends on the application. The question of keeping or removing stopwords is very frequent in NLP problems. More importantly, the choice of the list will affect the prediction power of the model. Deleting "non useful" words from the model will reduce dimension, but also could the model not understand underlying relationships correlated to these words.

1) Twitter stopwords: Using wordcloud vizualisation of most frequent words in Tweets, we can have a first insight of usefull and unusefull words. Before preprocessing, we can observe that TWitter specific words appear very frequently, for example "user" or "rt", which means "Re-Tweet", the action of sharing back an existing Tweet. We have chosen to delete these words that don't convey information about the feeling of the Tweet.

2) Common English StopWords: Famous NLP Python libraries such as NLTK, provide users with StopWords lists, containing a selection of "non-significant words". It appears that some words in the list could be very significant in order to classify positive and negative Tweets. Indeed, all the negative forms of verbs ("hasn't" vs. "has") give useful information of the polarity of a phrase. Moreover, prepositions and adverbs ("not", "nor", "any") are also useful in this situation. We therefore have modified the existing NLTK English StopWords list.

C. Text Normalization

The main goal of the text normalization is to keep the vocabulary small, which could improve the accuracy of our predictions, and to improve the time of training.

1) Lemmatization: Lemmatization considers the morphological analysis of the words and converts the word with meaningful way into the radical noun. Lemmatization has the advantage of keeping the meaning of words. We have used the Lemmatizer provided by NLTK library: WordNetLemmatizer.

2) Stemming: Stemming usually refers to a process of chopping off the last few characters.

Stemming can have bad effects for prediction. Overstemming comes from when too much of a word is cut off, and therefore is not accurate. Understemming comes when the word is not cut enough. We have tested two different stemmer algorithms. Porter Stemmer removes the common endings to words. Snowball stemmer is almost universally accepted as better than the Porter stemmer.

D. From text to trainable arrays: vectorisation

In order to make text understandable for the machine, we need to create a mapping between the words in the vocabulary list we chose, composed of all unique words in training data set reduced from the stopwords list. The simple solution for vectorization is a co-occurence matrix. However, it is useful for better predictions to create a mapping based on the meaning or occurrences of words. Therefore, two words with similar meaning or usage will be close. We have used GloVe Embedding method and RoBerta Transformer to achieve this using pre-trained models.

1) TFIDF Matrix: A TFIDF matrix is a numerical representation of a collection of documents that reflects the importance of each word within and across the documents. It is calculated by multiplying the term frequency (number of times a word appears in a document) by the inverse document frequency (logarithmically scaled inverse fraction of documents containing the word). The resulting TFIDF value reflects both the importance of the word within the document and its importance across the collection of documents. The matrix has a row for each document and a column for each word, with the value in each cell representing the TFIDF value for that word in that document. This way of representing trainable data has been used to train our sklearn models (i.e. logistic regression, descision tree and the multi-layer perceptron classifier)

2) N-grams vectorization process: N-grams vectorization process is based on a simple co-occurence matrix. However, this methodology adds N-grams, continuous sequences of

words or symbols or tokens in a document. N-grams vectorization process is only used with the TFIDF matrix. Hence, we used it in all our models using TFIDF matrices, and it improved our accuracy. This can help understand complex grammatical forms, such has "has not", that conveys a negative idea. This meaning is not understandable without using sequences of 2-3 words.

3) Padding sequence tokenizer: Padding is a technique used in natural language processing (NLP) to handle input data of varying lengths. It involves adding extra elements (such as zeros or special tokens) to the input data so that all the input sequences have the same length. Padding is often used when working with neural networks, as they require fixed-sized inputs. The added elements are typically placed at the beginning or end of the input sequence, depending on the padding strategy being used. Padding is an important step in preprocessing data for NLP tasks, as it allows the model to more easily learn patterns and relationships in the data. This technique allows neural network to capture the context of each tweet whereas the above TFIDF technique can't. Padding sequence representation of data has been used with our bidirectional LSTM model.

4) BERT and BERT-variant transformers tokenizer: BERT, or Bidirectional Encoder Representations from Transformers, is a natural language processing model that produces state of the art results in a large range of NLP tasks. BERT uses a specific tokenization method that takes into account the structure of the input language and is designed to preserve the meaning of the text as much as possible. The resulting tokens are then fed into the BERT model for further processing. We will use this BERT Tokenization to fine tune Transformers from Hugging face.

E. Principal Component Analysis

The PCA is used to reduce the number of drivers by keeping only the principal components directions. The dataset indeed contains thousands of different words, therefore implicating high dimension. We have tries several values of drivers. The PCA methodology reduces the accuracy but accelerates the process of training.

F. Comparing preprocessing methods

1) Approch: We compared the impact of each preprocessing method by comparing two logistic regression models, one with data preprocessed with the given method, the other without. If the method allows a better accuracy score, we keep this method and apply it to the following tests. We first tested the data cleaning, then the stopwords removal, the normalization methods and finally the vectorization. This methodology does not take into account all the possible combinations, but ensures to measure the impact of each method.

2) Results (on TABLE I): We finally chose to keep the methods with the label "With".

IV. Models and method

All the results of these methods are listed on table III.

Method	With	Without	Choice
Special characters	79,3%	79,0%	With
English stopwords	78,3%	79,0%	Without
Lemmatization	78,0%	78,4%	Without
Porter Stemmer	78,4%	78,4%	With
Snowball Stemmer	78,0%	78,4%	Without
2-Grams	79,5%	78,0%	With
3-Grams	78,3%	78,0%	Without
PCA 10 000 drivers	70,6%	78,0%	Without

TABLE I
COMPARING PREPROCESSING METHODOLOGIES

A. Logistic regression

Logistic regression is the first model used for the classification of observations. In statistics, logistic regression is a binomial regression model. We have used gradient descent to find the best parameters to fit the training data. Implementation of this method can be found on the sklearn_predictions.ipynb notebook.

B. Descision tree

We thought that random tree could be a good classifier for our tasks. After some optimization of the parameters, and taking into account the training time that augment very fast with the depth of the tree, we chose a depth of 100. Implementation of this method can be found on the sklearn_predictions.ipynb notebook.

C. Simple Multi-layer Perceptron classifier

A multi-layer perceptron (MLP) classifier is a type of artificial neural network that is used for supervised learning tasks, such as classification. It is called a multi-layer perceptron because it consists of multiple layers of interconnected "neurons," which are mathematical units that process input and generate output. We decided to implement it with two hidden layers of sizes 30 and 20. The activation function used is relu. Implementation of this method can be found on the sklearn_predictions.ipynb notebook.

D. Bidirectional LSTM Model

Bidirectional Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is trained to process sequences of data in both forward and backward directions. It is a variant of the standard LSTM network, which processes sequences of data in only one direction (either forward or backward). The main advantage of using a bidirectional LSTM is that it can capture both past and future context, which can be useful for tasks such as language modeling and machine translation. For example, in language modeling, it is often important to consider both the words that come before and after a given word in order to properly predict the correct word. Similarly, in machine translation, it can be helpful to consider both the source language and the target language in order to generate a high-quality translation. This time, using padding sequences and modeling from keras library, we trained a neural network summarised on TABLE II. Implementation of this method can be found on the bidirectional-LSTM_predictions.ipynb notebook.

Layer (type)	Output shape	Activation	# params
text (InputLayer)	[(None, None)]	None	0
embedding$_1(Embedding)$	(None, None, 128)	None	1280000
bidirectional$_1(Bidirectional - LSTM)$	(None, 200)	tanh	183200
dropout$_1(Dropout)$	(None, 200)	None	0
dense$_2(Dense)$	(None, 30)	relu	6030
dense$_3(Dense)$	(None, 1)	sigmoïd	31

TABLE II
BIDIRECTIONAL-LSTM MODEL SUMMARY

E. Fine tune transformers pre-trained models from HuggingFace

Fine-tuning a transformer is a process of adjusting the pre-trained model on a new dataset for a specific task. Transformers are a type of deep learning model that have achieved state-of-the-art results on a wide range of natural language processing tasks, such as language translation, language generation, and text classification. Fine-tuning a transformer involves using the pre-trained model as a starting point and adjusting the model's parameters on the new dataset in order to optimize it for the specific task at hand, which is here for our twitter dataset. Transformers library from hugging face provide objects and function scpecially designed to fine tune transformers models from a large range of BERT variant models. Here we have decided to fine tune the following pre-trained models from Hugging face :

- 'bert-base-uncased' (i)
- 'roberta-base' (ii)
- 'ProsusAI/finbert' (iii)
- 'distilbert-base-uncased-finetuned-sst-2-english' (iii)
- 'finiteautomata/bertweet-base-sentiment-analysis' (iv)

These pre-trained models, after fine tuning, gave various performances reported on table III. The BERT model (i) is the first introduced model of transformers, from google in 2018. Then, RoBERTa (Robustly Optimized BERT pre-training Approach) is an upgrade of the BERT model by Meta (ii), which is more robust and faster to train. Finbert (iii) is also a variant of BERT transformers, but with much less parameters and hence even faster to train, but it gives as we will see less good results. And finally the last model (iv) on the list is a BERT based model we found that was trained specifically on tweets for sentiment analysis. We will see that it gaves us from far the best accuracy. Implementation of this method can be found in the fine-tune-BERT_predictions.ipynb notebook.

V. RESULTS

TABLE III shows the results to compare different models test accuracy. Finally, we decided that the "finiteautomata/bertweet-base-sentiment-analysis" pre-trained transformers model was the best for our task and submitted it to AICrowd. It gave us a **86.7%** accuracy score for Submission #207476, by Jdbrs account.

Model	Test accuracy
Logistic regression	80.8%
random trees	76.6%
MLP classifier	81.1%
Bidirectional-LSTM model	83.4%
bert-base-uncased	83.0%
roberta-base	83.4%
ProsusAI/finbert	78.6%
distilbert-base-uncased-finetuned-sst-2-english	77.4%
finiteautomata/bertweet-base-sentiment-analysis	**88.0%**

TABLE III
COMPARISON OF MODELS

VI. SUMMARY

A. Conclusion

In fine, after testing several methods, we realized that the most important thing to have a good model is the text data tokenizing method. Methods of data vectorization that allows to best catch the context of a tweet naturally performs better. Then, the model has to be trained on the right data. As we can see, predicting tweets classification with BERT models trained on wikipedia underperforms a BERT variant model trained on tweets.

B. Possible improvements

Because we runned out for time, we didn't fine tuned other pre-trained models from Hugging face. But some of them were adapted to twitter data and hence much more adapted for these tasks than BERT and RoBERTa, that have been trained initially on wikipedia data. For example the pre-trained model "cardiffnlp/twitter-roberta-base-sentiment-latest" seems very good. During fine-tuning, because of a lack of experience, we didn't proceed to an optimization of hyperparamters, that could have certainly improved accuracy. Also, fine tuning is a very long training process and so we couldn't fine tune our models on all the tweets we had, but only on a very small portion of them (20'000 maximum) because it takes too much time, or because google colab can't handle too much data on a notebook session. Finally, it is possible to fine tune transformers models by considering transformers only as the "first layer of a bigger neural network". Maybe implementing this technique could give us better results.