Report NLP Project - Sentiment analysis

The aim of this report is to explain how was realized the final solution, from initial dataset to final pipeline. Every details would not be exposed since all cells of the notebook implementing the pipeline are commented and explained.

This project can be found on this repository : https://github.com/Proton013/NLP_Sentiment_Analysis_Project

And the

Table of Contents

Report NLP Project - Sentiment analysis

Table of Contents

I. Context

II. Data

a. Initial data

b. Data pre-processing

III. Solution : classifiers and RNN models

b. TF-IDF

c. LSTM

d. LSTM GloVe

IV. Solution : Pipeline

V. Improvements

I. Context

For a company, it is essential to understand well its clients. It is not always obvious too know what client think of our company. Nowadays, thanks to social medias, users give their feel freely.

A good application is to use those data to take the temperature of social medias about your company. The goal is to know whether your clients speak well or bad of your company and your products. This temperature can then be seen as an indicator to improve through communication campaigns or improvements of offers or products.

The application is rather simple, all we need is to train a sentiment analysis model. For each tweet, our model will need to be able to determine the sentiment of the person who wrote it.

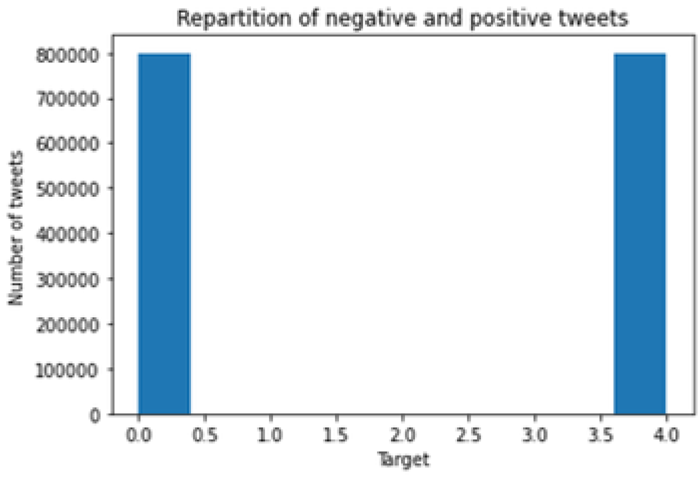
The main goal is thus the implementation of a pipeline that will take a tweet (or other text that holds a sentiment) and return positive or negative as a prediction.

To do so, I use google drive as a main storage directory for my models, datasets and scripts.

II. Data

a. Initial data

The initial dataset from Kaggle is used as a base for the different model's training and contains around 1,599,999 english tweets. Those are negative (1) and positive (4) tweets in equal proportion :



The initial labelled data has 6 label : target, time, date, query, username, text. We will only use target and text as input data for the training or the pipeline. A DataFrame is created holding only what is needed. There only are 2 classes target, meaning that in this dataset there is not neutral comments, only fully negative or fully positive tweets.

Since the text data is from twitter, special words can be found such as mentions (@), hashtags, unknown words, special characters, abbreviations, contraction, punctuation and other. Consequently, in order to efficiently work on this data, the tweets has to be processed before considering using them in the models.

b. Data pre-processing

To use the data as input in models, we need a representation of the tweets that wan be understood by the computer. The main idea is then to convert those tweet into lists of processed and sorted words. It will enable us later to have matrix representation of those tweets (float or integers).

The processing of the data follows few steps :

- pre-clean for contractions, mentions, hashtags, and other special characters/words ;
- pre-clean with known abbreviations replacement with their actual meaning ;
- tokenization of the pre-cleaned tweets ;
- after-clean of the resulting tokens :
    - removing of stopwords ;
    - stemming ;
    - lemming.
- dropping of empty token lists.

After trying the different function on a single tweet and making sure those are working properly, this pre-processing is done on the whole dataset. This processing drop the number of tweets from 1,599,999 to 1,588,504 which is negligible loss. The processing of such high number of tweets needs an awful long computation time with my implementation, so to gain time the pre-processed data is store into a google drive directory as a csv file. The processing only needs to be done one time, if we forget the scripts modifications and errors, and processed data cans then be loaded to the notebook for use.

For example, this tweet will become the following list of tokens :

```
'is upset that he cannot update his Facebook by texting it and might cry as a
resultSchool today also. Blah'

[['upset',
  'updat',
  'facebook',
  'text',
  'might',
  'cri',
  'resultschool',
  'today',
  'also',
  'blah']]
```

Afterwards, because the dataset is too big for efficient calculations and RAM uses (may crash the notebook otherwise), it length is reduced to 4000 tweets still in equal sentiment proportions by taking the first 2000 and the last 2000.

Finally, the target is replaced to be in a binary case : 4 becomes 1, which give 0 for negative and 1 for positive.

III. Solution : classifiers and RNN models

For now, 2 classifiers and 2 RNN models are computed and trained to make the sentiment prediction on a given tweets :

- **BOW** : uses the bags of words of a tweet ;
- **TF-IDF** : uses Term Frequency-Inverse Document Frequency of a tweet, giving more information on each words as floats ;
- **LSTM** : Long-Short Term Memory ;
- **LSTM GloVe** : Long-Short Term Memory with use of the Global Vectors for Word Representation dataset.

a. BOW

This first classifier, a Decision Tree, takes as inputs the tokens in a bag of words representation. After calculating the vocabulary vector of the model input dataset (in our case reduced to 4000 tweets), the frequency of each word is computed into a vector of the same length. It gives a first representation that does not hold great meaning but still.

At the end of the classifier fit, we obtain a mean accuracy of 0.648 when tested with the validation data. The prediction is quickly given and with a fit on a greater amount of data, its accuracy could be increased.

b. TF-IDF

This second classifier, a Random Forest, takes as inputs the TF-IDF (Term Frequency-Inverse Document Frequency) representation of the processed tweets. In comparison with the above BOW classifier, this statistic metrics allows to evaluate the significance of each word in a document relatively to a collection of documents. The weights (float between 0 and 1) increase proportionally to the number of occurrences of a word in all documents and also variates according to the word frequency.

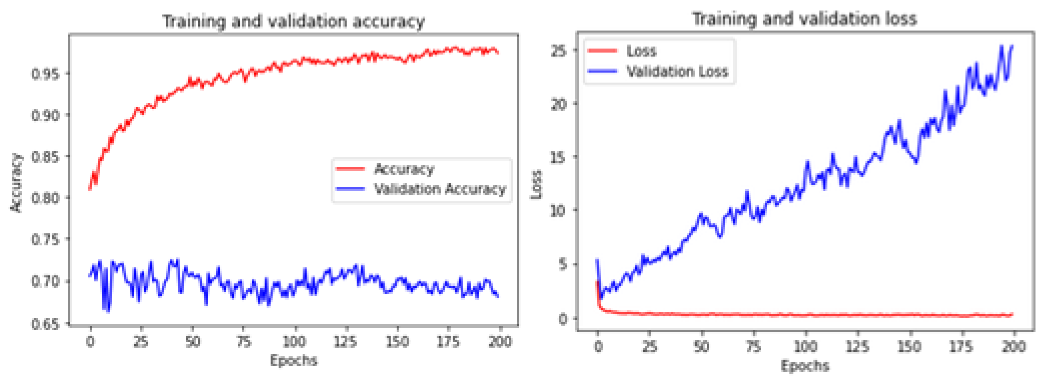
The TF-IDF representation is calculated through a TF-IDF converter fitted on the reduced input data, and this representation gives greater meaning to the vector and thus a better prediction. The accuracy of this classifier is 0.66875 and same as the BOW classifier it could be increased with more training on more data.

c. LSTM

This LSTM model is a RNN model that has feedback connections. It is known that LSTM models are more effective than Deep Neural Networks and conventional RNNs for sentiment analysis. In fact, LSTM (Long-Short Term Memory) save the words and predict the next words based on the previous words. LSTM is a sequence predictor of next coming words.

Using a Tokenizer fitted on the reduced input data, the tokens are converted to padded sequences to an arbitrary maximum length. The resulting representation is a float matrix used as input in the model.

This model is composed of Embedding and LSTM layers and gives an accuracy of 0.6808 after a fit of 200 epochs and of batch size 80. The training can be resumed to the following graphs :

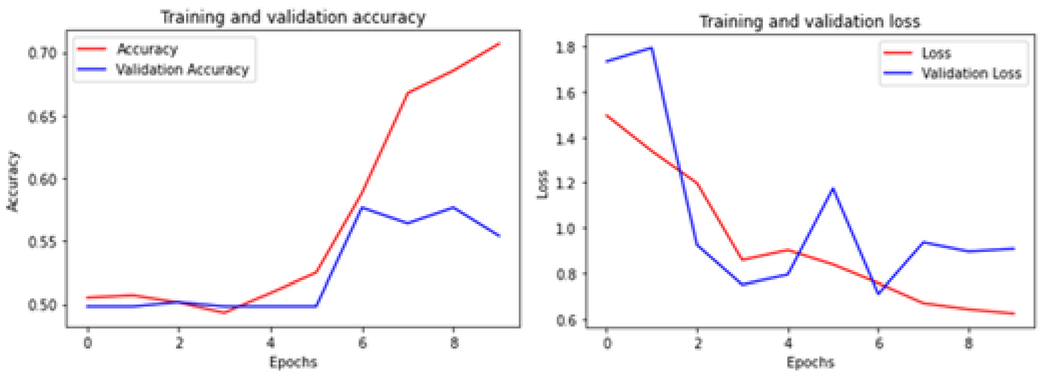


d. LSTM GloVe

The idea is the same has the LSTM model since it uses the same LSTM layers, but in addition GloVe Embedding (Global Vectors for Word Representations) layer is used. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. It gives us in this manner a more meaningful representation of the tweets.

GloVe dataset makes its entrance in the definition of the Embedding layers used in the model, computing the weights of the layers in a matrix.

Apart from this Embedding layer, this model inputs and operation are the same. An accuracy of 0.5725 is reached with 10 epochs with a batch size of 64. No further training were due to previous unsuccessful attempts of training. Indeed, a very low accuracy was reached with the previous attempts and the reasons were nowhere to be found. In any case, the training could be resumed as follow :



IV. Solution : Pipeline

The final pipeline implements all 4 models into one function and does the need processing on the input tweet. The pipeline function takes as another argument the name of the model to use. In fact, this pipeline enables the user to compare the accuracy and time consuming of each model.

About its operation, it processes the tweet to clean and tokenize it and depending on the chosen model, it calculates or load from google drive directories the needed parameters to use the models. Those return a predictions, a float between 0 and 1 (or simply 0 or 1) that is rounded and interpreted as negative or positive.

The YELP API is also used to test the pipeline : a request to random business is send and a random review is collected from the response.

V. Improvements

Many improvements could be done on this pipeline solution :

- the pre-processing could be more efficient and optimized to take less time with a more intelligent cleaning ;
- better models could be implemented and added to the pipeline ;
- existing models could be trained on other set of data (within the initial Kaggle dataset or other) and optimized, especially the RNN models and their layers ;
- use an more adequate API to test a tweet analysis sentiment pipeline like the Twitter API ;
- implement other models for different languages since for now, only English is available.

Of course, some improvements are more easy than other but all can be achieved in some time.