Comparing Microsoft Azure Machine learning services to perform sentiment analysis on tweets.

In this article, I will compare three different methods to perform sentiment analysis on a corpus of text containing 1,600,000 tweet using the Microsoft Azure Services. One of them will be by using directly the Azure API and the other two by building models using machine learning studio (drag and click) and machine learning Software Development Kit for programming.

Azure cognitive service API

Azure offers an API for multiple text analysis whether you want to be able to detect language, identify key ideas in a text or detect relative sentiment of said text. A total of 4 methods can called while using the API and we will focus on the one that is interesting for the scope of this project: sentiment analysis. However, keep in mind that for any subscription you have access to all four methods.
The Sentiment analysis methods return three coefficients between 0 and 1 corresponds to the following value for the text: negative, neutral and positive. You also get a confidence score allowing to estimate how well has the estimation been done by the algorithm.
With a free subscription, you are limited to 10 texts per request. The first level of subscription allow you to do 200 requests per second and 300 total per minute.
Regarding performance the sample tested gave an accuracy around 0.7, which is not to be taken as a full test since the whole dataset was not tested.

Building model

Data

During the whole experiment, I used the same split for the data that was as follow:
80% training, 10% validation and 10% for test. Every metrics mentioned in the rest of this article concern metrics measured on the test set.

Metrics

Accuracy was used during the experiment mainly because it is easily interpretable, we had a balanced dataset and both output (negative or positive sentiment) were of equal importance.

Azure Machine Learning Studio

The next features tested was the use of the Machine Learning studio to build, train and test a simple model. The interface is of a “drag and click” type, which make it quite user friendly to build the model and visualize the pipeline. For any block, you can choose among a reasonable amount of features, which allow the user to fit the model to the need of the project within a certain limit. For this experiment, we build a simple logistic regression preceded by a block of pre-processing text to clean the tweet. The main features were chosen to perform the following: delete mail address, tweet address, hashtag, punctuation and number along with a stemming layer. The text pre-processing block was followed by a bi-gram vectorization before feeding the data to the logistic regression.

After training the dataset and tuning a few parameters this model gave an accuracy of 7.78. This level was reach surprisingly fast considering the size of the dataset. Training time was around 2-3h depending on the availability of the cluster compute power from Microsoft.

Advanced Model

For the third step of this experiment, we coded every block of the script and run the different experiment using the Azure SDK.

Preprocessing
The first script was dedicated to the pre-processing of the tweet in order to clean the text and get it ready to be vectorized. We took almost the same step as for the customized model but specific features were added to treat the slang and other language abuses that one can find on twitter. Namely, function that turns known slang into correct sentence. Example: yall nice! ➔ You are all nice.
Repeating more than three time a single character were turn into a three time characters to flag the use this emphasis. Whaaaaaaaat =➔ whaaat. Which make a difference later in the vectorization with a regular what.
We also tested two different methods regarding punctuation: one that kept track of emoji using the *TweetTokenize* function from NLTK and one that simply delete all punctuation.

Vocabulary size
By deleting any word, occurring less than twice in the corpus the method without punctuation had a vocabulary size of about 56,000 and the methods with the emoji has a vocabulary size of about 57,000

Models
Three different models were tested one logistic regression to be compared with the customize model, one neural network with a simple RNN layer and one neural network with a LSTM layer. We then tested the best model with a pre-trained embedding layer from GloVe. Both Neural Network had the same construction to be able to compare the performance between simple RNN and LSTM, which was as follow:

- Embedding layer
- RNN or LSTM
- Dense
- Drop out
- Dense

For the logistic regression, we fixed the number of iteration to 200 and run several test on the validation test to identify the correct c-parameters.

For the neural network, we ran the test on 100 epoch and used a batch size of 128.

Performance:
The pre-processing method including emoji performed in general better no matter the model. The simpleRNN model performed the worst in the experiment with an accuracy slightly above 0.7. We reached an accuracy of 7.78 with the logistic regression and around 0.79 with the LSTM neural network.

Pre-trained embedding:

The best performance was reached by using the pre-trained embedding matrix from GloVe. They released a specific version for twitter, which performed very well with LSTM. We had to modify the pre-processing to fit the embedding matrix. We removed stemming, let the punctuation and and repeating letters. We manage to have 92,84% of our corpus which was embedded in the GloVe dictionary which covers 99,59% of all texts. Flags and emoji were included in the GloVe matrix.