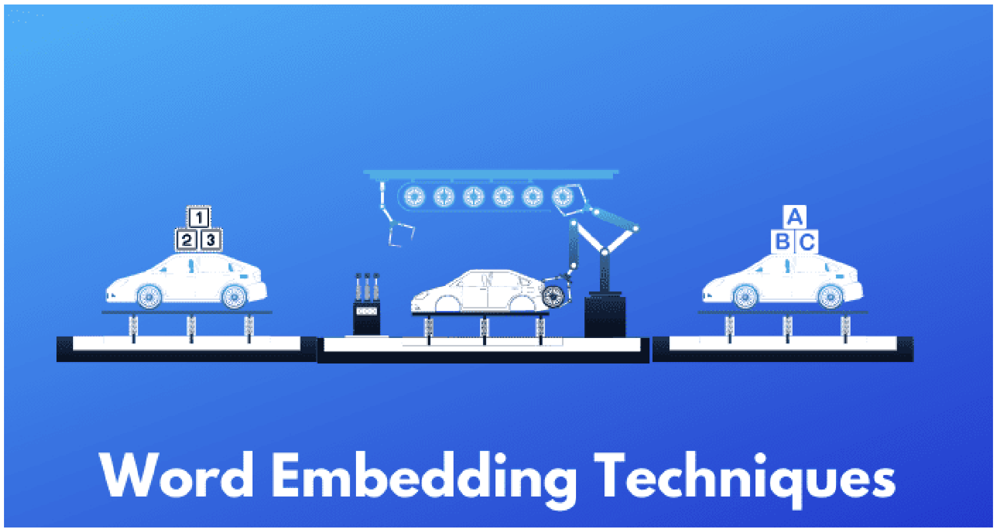
Comparing Different Approches for Sentiment Analysis



In this globalized and hyper connected world, social networks have become part of our lives. Not only do we share images and articles, but we can also question services or share bad experiences. That's why companies want to avoid all the "bad stuff" that can affect their company's image. To achieve this, you would need to be able to detect NEGATIVE messages on the Internet so that you can act before word spreads.

Sentiment Analysis is one of the most classic Natural Language Processing (NLP) problems : **Given a piece of text, would you say its rather *POSITIVE* or *NEGATIVE* ?**

Although it seems almost natural for a human mind to classify simple sentences, there are multiple challenges that can make this task much more complicated:

- **language**: the given sentence could be in any language, potentially one you don't understand
- **language quality**: even if you know the language, the sentence could be written in a very un-intelligible way (with spelling, conjugation, grammar, syntax errors, ...)
- **language technique**: even in a perfectly well written English, the author could use a rhetorical device to imply a different meaning than the literal sense of the words (humor, derision, irony, sarcasm, ...)
- **context**: taking a sentence out of its context can completely change its meaning
- **subjectivity**: different people will interpret the same sentence differently depending on their personal way of thinking

In this article, we are going to cover a comparison of the three approaches ("Simple Custom Model", "Advanced Custom Model" and "Advanced BERT Model") that we can use to predict the sentiment of tweets. But first, let's start with the exploratory data analysis (EDA)

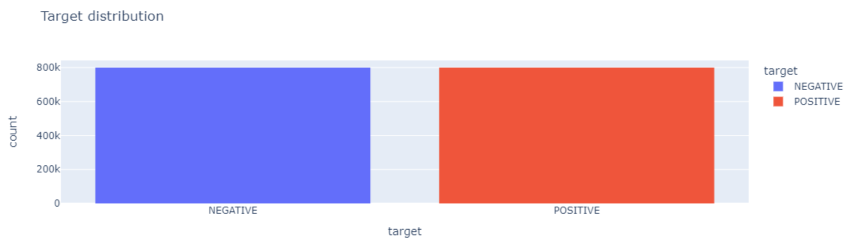
Exploratory Data Analysis

In this section, we are going to perform an EDA to understand the *text* and *target* variables. The data we are going to use is Kaggle - Sentiment140 dataset :

- **text** : 1.6 million tweets
  - low language quality: many Twitter specific words ("RT", @username, #hashtags, urls, slang, ...)
- **target** : binary categorical variable representing the sentiment of the tweet
  - 0 = *NEGATIVE*
  - 4 = *POSITIVE*

Target variable

Let's have a look at how the *target* variable is distributed. The *target* variable is perfectly **balanced**:



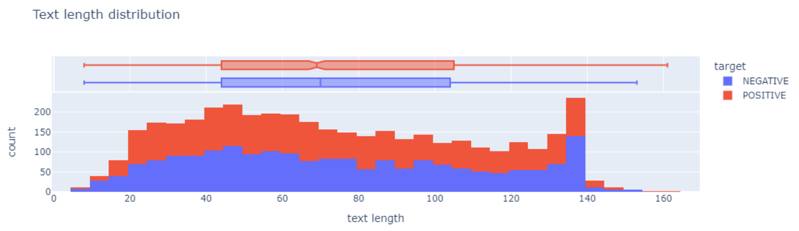
There are exactly as many (800000) POSITIVE tweets as NEGATIVE tweets. There are no NEUTRAL tweets. The problem is well balanced and there will be no bias towards one class during the training of our models.

Text variable

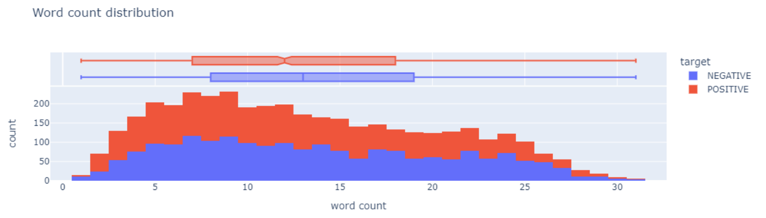
Text length

Let's have a look at what the text variable looks like. POSITIVE tweets are slightly (not significantly) longer than NEGATIVE tweets. In both classes, there are two modes:

- ~45-50 characters and 135-140 characters (the maximum allowed at the time the data was extracted):



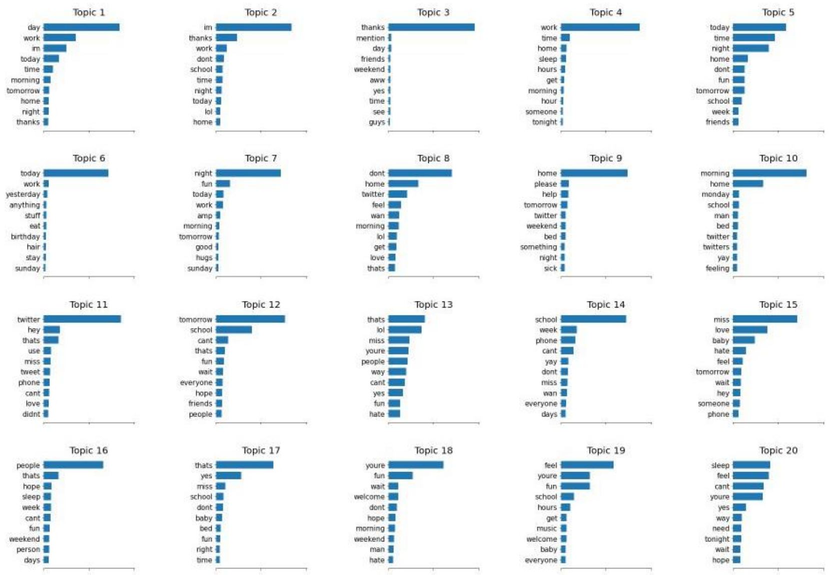
- *~7 words* and *~22 words* :



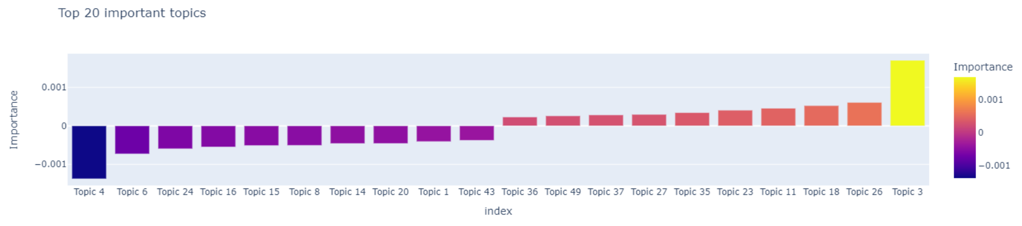
Topic modeling

Let's see what **topics** (group of words frequently found together) are **important** in the *text* variable.

Running a LSA on the cleaned text, we can identify **topics** :



Running a simple Logistic Regression, we can measure the importance of each topic towards the target variable :

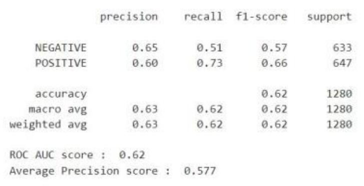


Word embeddings
In order to map our speech and texts to numerical form, however, we want to select a method where the semantic relationships between words are best preserved and for numerical representations to best express not only the semantic but also the context in which words are found in documents.
Word embeddings is a special field of natural language processing that concerns itself with mapping of words to numerical representation vectors following the key idea – "a word is characterized by the company it keeps".
To generate word representations, one can use methods like bag of words or term frequency-inverse document frequency (TF-IDF). In this article, we will however focus on word embedding methods which are more advanced in nature.

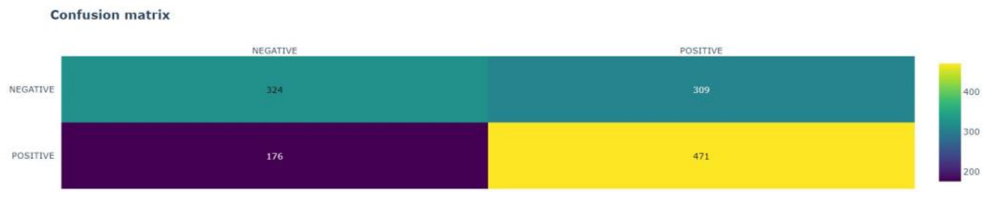
Protocol
To be able to run the our dataset using the artificial neural network named Long Short Time Memory (LSTM), we split our dataset into a *train, Validation* and a *test* datasets, and compare the classification results according to different binary classification metrics using the validation dataset:

- **Precision** : measures how many observations predicted as positive are in fact positive.
- **Recall** or **Sensitivity** : measures how many observations out of all positive observations have we classified as positive.
- **Specificity** : measures how many observations out of all negative observations have we classified as negative.
- **Accuracy** : measures how many observations, both positive and negative, were correctly classified.
- **F1-score**: combines *Precision* and *Recall* into one metric.
- **Average Precision (AP)**: average of precision scores calculated for each recall threshold.
- **ROC AUC** : tradeoff between *True Positive Rate (TPR)* and *False Positive Rate (FPR)*.
- **Confusion Matrix** : common way of presenting *True Positive (TP)*, *True Negative (TN)*, *False Positive (FP)* and *False Negative (FN)* predictions.

Word2vec
Word2vec is based on a shallow, two-layer neural network, which takes as input a corpus of texts and produces as the result a vector for each word that is represented in the corpus.
The key feature of the shallow neural network that is learned is – words that often occur in similar contextual locations in the corpus also have a similar location in the word2vec space. As words that have similar neighbour words are likely semantically similar, this means that the word2vec approach is very good at retaining semantic relationships.
There are two major Word2vec embeddings methods. While typical machine translation models predict the next word based on prior words, word embeddings are not limited in a similar way. Word2vec authors thus used both n words preceding the target word as well as m words after the target word in an approach known as continuous bag of words or CBOW approach. A second approach to Word2Vec is called Skip-Gram model and is based on predicting the surrounding words from the current word.
One can also use a pre-trained Word2Vec models, with the most popular one is a word vector model with 300-dimensional vectors for 3 million words and phrases. The model was trained on part of the Google News dataset (about 100 billion words). To work with our dataset, we used this approach. We download the pre-trained model with 300 dimensional vectors. In the following figure, you can find the binary classification metrics of the validation dataset.



```
              precision    recall  f1-score   support

    NEGATIVE       0.65      0.51      0.57       633
    POSITIVE       0.60      0.73      0.66       647

    accuracy                           0.62      1280
   macro avg       0.63      0.62      0.62      1280
weighted avg       0.63      0.62      0.62      1280

ROC AUC score :  0.62
Average Precision score :  0.577
```

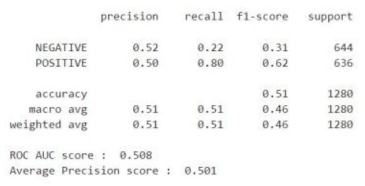


GloVe

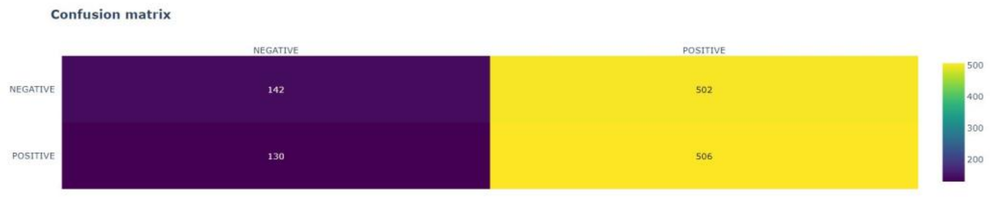
GloVe is also a very popular unsupervised algorithm for word embeddings that is also based on distributional hypothesis – "words that occur in similar contexts likely have similar meanings".GloVe learns a bit differently than word2vec and learns vectors of words using their co-occurrence statistics.

One of the key differences between Word2Vec and GloVe is that Word2Vec has a predictive nature, in Skip-gram setting it e.g. tries to "predict" the correct target word from its context words based on word vector representations. GloVe on is however count-based.

It first constructs a matrix X where the rows are words and the columns are contexts with the element value X_ij equal to the number of times a word i appears in a context of word j. By minimizing reconstruction loss, this matrix is factorized to a lower-dimensional representation, where each row represents the vector of a given word. Word2Vec thus relies on the local context of words, whereas GloVe utilizes global statistics on word co-occurrence to learn vector representations of words. To work with our dataset, we download the pre-trained model with 300 dimensional vectors. In the following figure, you can find the binary classification metrics of the validation dataset.



```
              precision    recall  f1-score   support

    NEGATIVE       0.52      0.22      0.31       644
    POSITIVE       0.50      0.80      0.62       636

    accuracy                           0.51      1280
   macro avg       0.51      0.51      0.46      1280
weighted avg       0.51      0.51      0.46      1280

ROC AUC score :  0.508
Average Precision score :  0.501
```

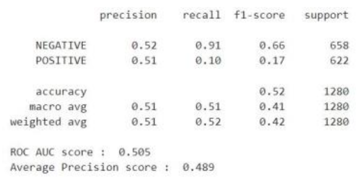


Fasttext

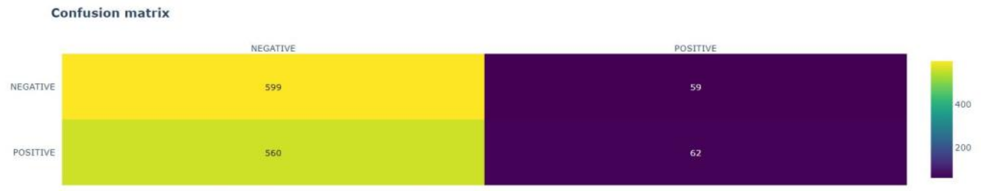
FastText was introduced by T. Mikolov et al. from Facebook with the main goal to improve the Word2Vec model. The main idea of FastText framework is that in difference to the Word2Vec which tries to learn vectors for individual words, the FastText is trained to generate numerical representation of character n-grams. Words are thus a bag of character n-grams.

If we consider the word "what" and use n=3 or tri-grams, the word would be represented by the character n-grams: <"wh","wha","hat","at">. < and > are special symbols that are added at the start and end of each word.

By being based on this concept, FastText can generate embedding vectors for words that are not even part of the training texts, using its character embeddings. As the name says, it is in many cases extremely fast. FastText is not without its disadvantages – the key one is high memory requirement, which is the consequence of creating word embedding vectors from its characters. To work with our dataset, we download the pre-trained model with 300 dimensional vectors. In the following figure, you can find the binary classification metrics of the validation dataset.



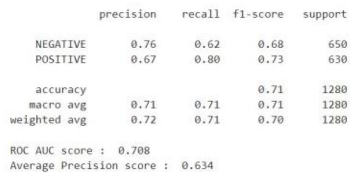
```
              precision    recall  f1-score   support

    NEGATIVE       0.52      0.91      0.66       658
    POSITIVE       0.51      0.10      0.17       622

    accuracy                           0.52      1280
   macro avg       0.51      0.51      0.41      1280
weighted avg       0.51      0.52      0.42      1280

ROC AUC score :  0.505
Average Precision score :  0.489
```



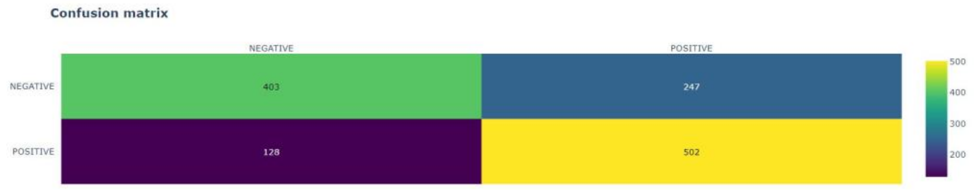
BERT

Hugging Face is an open-source library for building, training, and deploying state-of-the-art machine learning models, especially about NLP. Hugging Face provides two main libraries, transformers for models and datasets for datasets.

BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers. BERT was trained with the masked language modeling (MLM) and next sentence prediction (NSP) objectives. It is efficient at predicting masked tokens and at NLU in general, but is not optimal for text generation. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. To work with our dataset, we download the pre-trained model with 300 dimensional vectors. In the following figure, you can find the binary classification metrics of the validation dataset.



```
              precision    recall  f1-score   support

    NEGATIVE       0.76      0.62      0.68       650
    POSITIVE       0.67      0.80      0.73       630

    accuracy                           0.71      1280
   macro avg       0.71      0.71      0.71      1280
weighted avg       0.72      0.71      0.70      1280

ROC AUC score :  0.708
Average Precision score :  0.634
```



Conclusion

Machine learning algorithms expect the input data to be in a numerical representation. Texts can be encoded with a wide range of methods, from simple ones like bag of words approach (BOW) and good results can often be achieved with the TF-IDF method.

Further improvement in recent years was development of Word2Vec and GloVe, both based on distributional hypothesis or observation that words that occur in the same contexts often have similar meanings. FastText framework soon followed which allowed generating good vector representations for words that were either rare or did not appear in the training corpus. BERT just arrive to revolutionized the NLP field, moving forward in a hopeful direction.

Development of word embeddings as well sentence embeddings is an important subfield in the natural language processing, and we expect further important breakthroughs in this field in the near future as well as for it to play an important role in various NLP applications.