Challenge 3 : Tweet sentiment analysis

Groupe 50 : Maxime MICHEL, Matteo NOTTARIS, Nicolas BOINAY, Mathieu OLIVIER

Project Objective

- Analyse sentiments of customers in tweets.
- Help companies to understand better the reaction of their customers in their tweets.
- Our goal : Based on tweets from Figure Eight's Data for Everyone platform, classify them in three categories: ***Positive, Neutral, Negative***

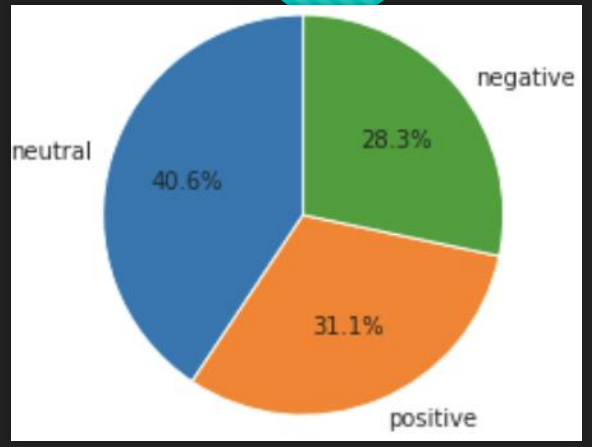
- We worked on two different methods : ***Bag-of-words*** **and** ***BERT Transformer***.

Process main steps

1. Data inspection and pre-processing
2. Bag of Words
3. Bert
4. Result analysis

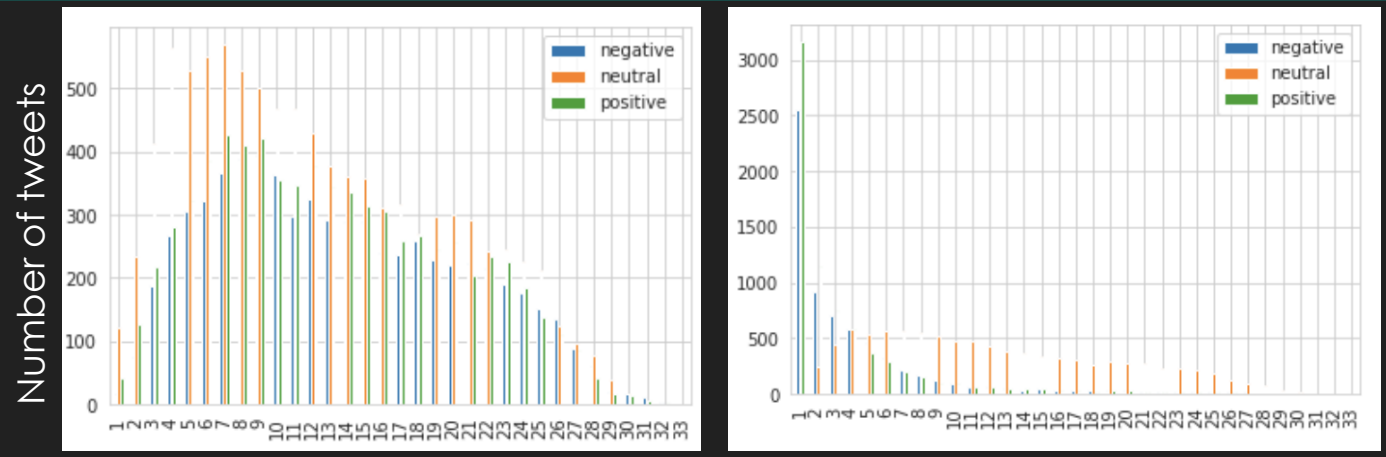
Data inspection

It seems that working with 'selected_text' instead of 'text' greatly improves the accuracy of our analysis. We'll therefore work with this feature for the moment.



Distribution of sentiments

The classes seem to be balanced. We don't have to do any data generation to balance them.



Correlation between number of words and sentiment

It appears that there is not any clear difference in the train repartition. We therefore think that we can't use the number of word as some kind of feature. In the test datasets most of the tweets are composed of 1 word, we'll keep that in mind.

```
Int64Index: 23495 entries, 17850 to 2732
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   textID         23495 non-null  object
 1   text           23495 non-null  object
 2   selected_text  23495 non-null  object
 3   sentiment      23495 non-null  object
 4   target         23495 non-null  int64
```

```
RangeIndex: 2748 entries, 0 to 2747
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   textID         2748 non-null   object
 1   text           2748 non-null   object
 2   selected_text  2748 non-null   object
```

No NaN or Null values in the train or test dataset !

Data Preprocessing

- **Let's clean the data** There is no messing data (no Nan ...) so we can proceed to the next step of our pre-processing : (adapted to NLP)
  - Remove none alphabetic characters
  - Make the word lower case
  - Remove the stop words
  - Stemming(process of finding the base word)
  - Spell correction

```
def cleaner(x):

    # remove non alphabatic characters
    x = re.sub('[^A-Za-z]', ' ', x)

    # make words lowercase (to reduce size of vocabulary)
    x = x.lower()

    # tokenising
    tokenized_x = wt(x)

    processed = []
    for word in tokenized_x:
        if word not in set(stopwords.words('english')):
            processed.append(spell1(stemmer.stem(word)))
    cleaned_text = " ".join(processed)
    return cleaned_text
```

The NLTK package was used to achieved this pre-processing.

Afterwards a simple train_test_split from sklearn to have a validation dataset and it's ready to be processed !

Bag-Of-Word

Bag of words is a NLP technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. We used the sklearn text processing package.
Two approaches were used :

1. The "CountVectorize()" provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. Because these vectors will contain a lot of zeros, we call them sparse. The vectors returned from a call to transform() will be sparse vectors

2. An alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF. The frequency of each token gives them a score. It allows to highlight words that are more interesting. We used "TfidfVectorizer()"

Scores obtained using a simple Naives Bayes Classifier :

Normal :

```
The accuracy of our multinomial Naive Bayes classifier is: 78.98%
The fbeta score is: 0.7878608013392294
```

TF-IDF :

```
The accuracy of our multinomial Naive Bayes classifier is: 78.09%
The fbeta score is: 0.7778661398131096
```

We see that there isn't any improvement using TF-IDF.
The results deeper analysis are done afterward.

BERT : Bidirectional Encoder Representations from Transformers

Like word embeddings, BERT is also a text representation technique which is a fusion of variety of state-of-the-art deep learning algorithms, such as bidirectional encoder LSTM and Transformers.

We have used "bert-base-cased" from the HuggingFace library and pyTorch.

Process :

1. Add special tokens to separate sentences and do classification (done using encode_plus())
2. Padding : Pass sequences of constant length
3. Create array of 0s (pad token) and 1s (real token) called attention mask
4. Creation of dataSet adapted to pyTorch and then dataLoaders
5. Use a dropout layer for some regularization and a fully-connected layer for our output (the rest is BERT)
6. Training and evaluating

Special Tokens :

- [SEP] - marker for ending of a sentence
- [CLS] - we must add this token to the start of each sentence, so BERT knows we're doing classification
- [PAD] for padding

Parameters :

- EPOCHS = 10
- Learning Rate = 3e-5
- Batch size = 16

Adapted Bert

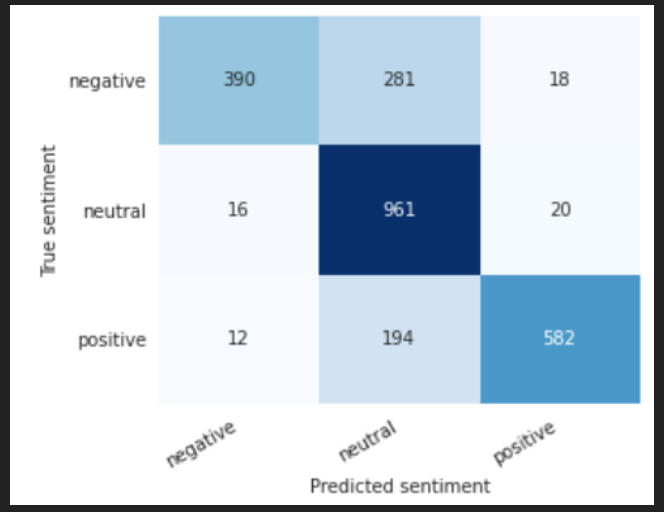
```
class SentimentClassifier(nn.Module):
    def __init__(self, n_classes):
            super(SentimentClassifier, self).__init__()
            self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
            self.drop = nn.Dropout(p=0.3)
            self.out = nn.Linear(self.bert.config.hidden_size, n_classes)
    def forward(self, input_ids, attention_mask):

        returned = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask
            )
        pooled_output = returned["pooler_output"]
        output = self.drop(pooled_output)
        return self.out(output)
```

The classes were renamed 0,1,2 instead of 0,1,-1 for compatibility with the model

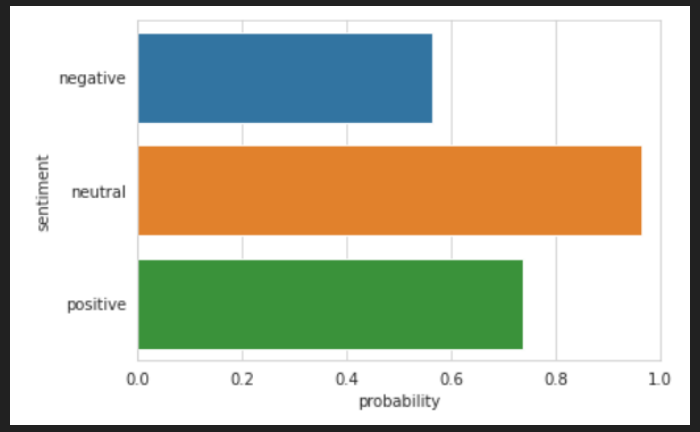
Results Analysis (On validation data)

Bag of Words

Confusion Matrix

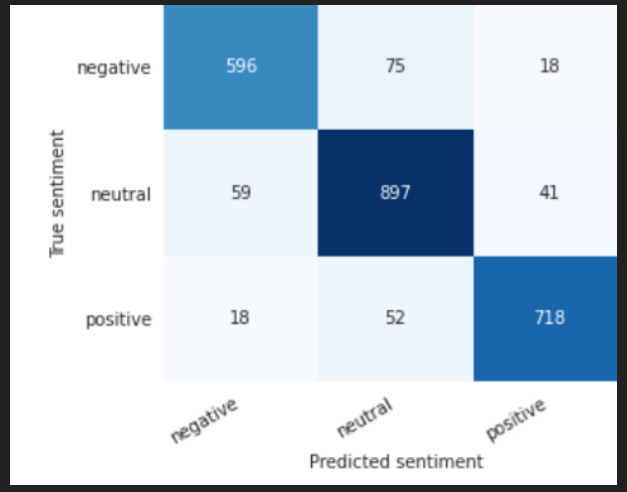


Confidence of each sentiment

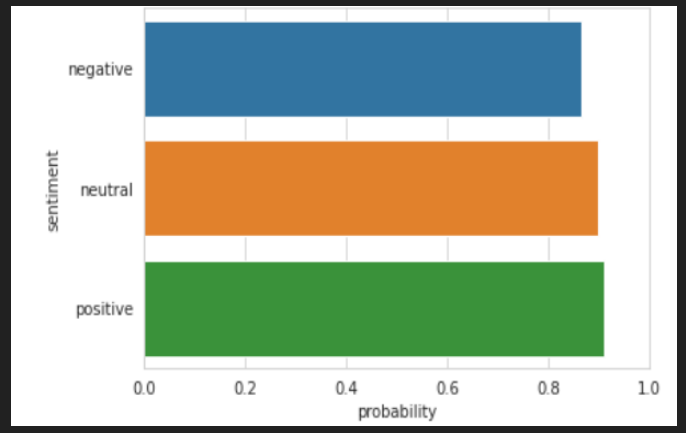


BERT

Confusion Matrix



Confidence of each sentiment



Analysis

Bow :

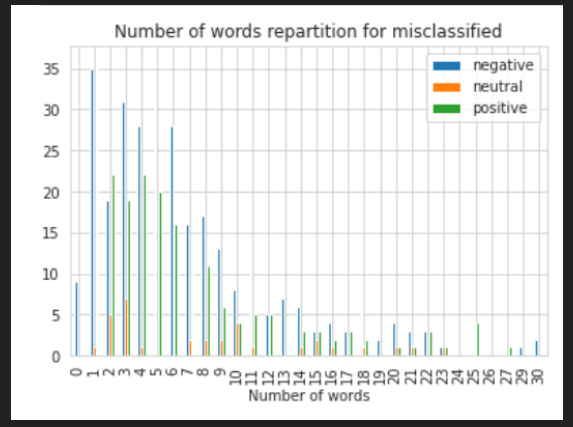
The neutral tweets are significantly better classified than the 2 others. It illustrates the tendency of this model to classify to neutral more than it should. **The model is biased to neutral !**

Bert :

No clear difference, it seems that every sentiment gives approximatly the same accuracy. **This model treats equally all sentiment**, which is a good sign.

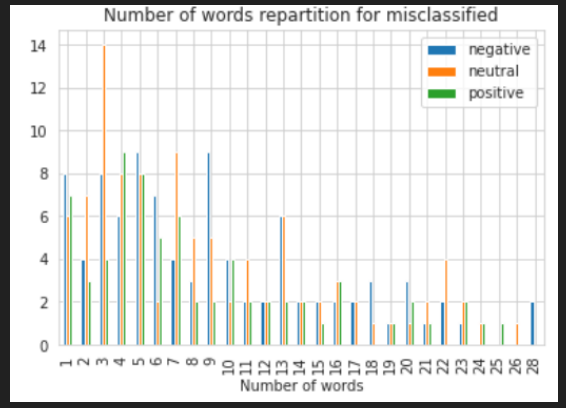
Results Analysis (On validation data)

Bag of Words



Best fbeta score : **0.773**

BERT



Best fbeta score : **0.892**

Analysis

It seems that shorter words are more inclined to be misclassified. But actually this is just because there are more short tweets than long. Therefore it seems that our models aren't that affected by the number of words.

The most important indicator of performance is the fbeta score. BERT gives way better results than BOW.

Let's focus on the one that aren't well classified and analyse the actual tweets (by our best model ie BERT) :

	selected_text	sentiment	predicted
6008	She didn`t make the challenge	neutral	negative
7451	****...the right side of my earphones just sto...	negative	neutral
2491	Ahh its so gloomy out I`m pretty sure I just ...	neutral	negative
6815	leaving	negative	neutral
13840	we didnt see you though	negative	neutral

As expected a major part of the tweets that are wrongly classified are even complex to classify for a human being. The given true sentiment are not obvious choices in all the cases.

« Leaving » stated as a negative tweet is not obvious and therefore we couldn't expect our model to be right in every of those cases.

Conclusion

- NLP is very complex and a right pre-processing is very important. Our models couldn’t understand some tweets written with strange typo.
- BERT transformers is a powerfull tool and showed its capacity to really overtake the Bag of Word method. Several BERT exists and we used the simplest but others should be tried to adapt to the particularity of the wanted task.
- Classifying tweets on a 3 sentiments base can include mistakes even when done by a human. The frontier between neutral and positive/negative can be blurry and therefore the model (as a human) can't really distinguish them. We tried ourselves to give the classes of some of the misclassified ones and in some case we were agreeing with the model. The question we could ask is whether the human or our model is actually wrong.