M2 AIC : Information retrieval project
Sentiment Classification

Wafa Bouzouita

Introduction

In this present project, our goal is to classify moovies reviews in two types of reviews: positive and negative. This technique is called sentiment analysis. To realize our objective, we will use three methods : Naive Bayes, SVM and convolutional neural network. we will then implement this methods and evaluate its performance. The data set that we used is constructed from IMDB.

In this project, we used 3 references [1], [2] and [3].

1 Data description

During this project, we are using the IMDB dataset. It is constituted of two files : one with positive reviews (imdb.pos) and one with negative reviews (imdb.neg). Each file contains 300000 reviews, one per line. we downloaded this data from the link.

2 Models

To build the model we proceeded in three steps :

1. Gathering perfect Data for training and testing.
2. Vectorizing the data.
3. Creating the Model to train and then predict.

2.1 Load and clean the data

The first part consisting in making data clean and prepared for training. To do that, we firstly, load the data by using the function *loadTexts* described in the notebook. The data have a big size, training all this data is time consuming. So, we take from the IMDB data 7000 positive reviews and 7000 negative reviews. The raw text is pretty messy for these reviews so before we can do any analytics we need to clean things up.
To clean this data, the function *clean_str* is used. This function uses regular expressions to lower the text, remove the spaces ... After loading and cleaning the data, we split it between two sets : train and test in the case of using SVM and Naive Bayes methods. However, in the case of using convolution neural network, we split the data in three sets : train, dev and test. For the two cases we used the function *train_test_split.* And we kept a 50/50 ratio between positive and negative instances in each set, by setting the parameter of the last function, *stratify* to 1. we also shuffled the data by using the parameter *random_state* to prevent overfitting. The data obtained for the SVM and Naive bayes, was splitted as follows :

- train data : 11200 reviews.
- test data : 2800 reviews.

Whereas for the CNN, data is splited as follows :

- train data : 11340 reviews.
- dev data : 1400 reviews.
- test data : 1260 reviews.

Stop words : In computing, Stopwords are words that are generally considered useless. Most search engines ignore these words because they are so common that including them would greatly increase the size of the index without improving precision or recall. However, problems like sentiment analysis are sensitive to stop words removal. In fact, using a generic list of stopwords (to remove) can have a negative impact on sentiment analysis performance. An example could be the following sentence: "I told you that she was not happy". When removing the stop words, the result would be [‘told’, ‘happy’]. For sentiment analysis purposes, the overall meaning of the resulting sentence is positive, which is not at all the reality.

N-grams : In computanional linguistics, n-grams represents a contiguous sequence of n items from a given sample of text or speech. In our type of problem, considering only single word features, which we call 1-grams or unigrams, doesn't give the best result. In fact, We can potentially add more predictive power to our model by adding two or three word sequences (bigrams or trigrams) as well. For example, if a review had the three word sequence "didn't love movie" we would only consider these words individually with a unigram-only model and probably not capture that this is actually a negative sentiment because the word ‘love’ by itself is going to be highly correlated with a positive review.

2.2 Vectorizing the data

For making sense to our machine learning algorithm we'll need to convert each review to a numeric representation, which we call vectorization. In this project, we will vectorize the data set using the approach of Bag-of-words.

2.2.1 Bag-of-words

The bag-of-words model is commonly used in methods of document classification where the occurrence of each word is used as a feature for training a classifier. So to create this representation model, we used the *sklearn* function *CountVectorizer* which 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.

2.2.2 Bag-of-words with TF-IDF

The bag-of-words representations that we have used does not taking into account the context of the corpus. A better approach would be to consider the relative frequency of tokens in the document against their frequency in other documents. The central insight is that meaning is most likely encoded in the more rare terms from a document. For example, in a positive review, tokens such as "excellent," "good," and "amazing" appear more frequently than in negative review.

TF–IDF, term frequency–inverse document frequency, encoding normalizes the frequency of tokens in a document with respect to the rest of the corpus. This encoding approach accentuates terms that are very relevant to a specific instance. To create this model, we used the *sklearn* function *TfidfVectorizer.*

2.2.3 Word embedding

Word embedding is a popular technique used to represente the document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words. We used this method to vectorize the inputed data, in the case of using CNN model. Where firstly, we constructed a dictionary that consider all the words contained in our training data (other words are considered unkonwn), where each key refer to the word, and each value refer to the index of that word in the given text document. After that, to each tweet in our training data , we applied the function *Embedding(num_ embeddings, embedding_ dim)* (taken from the *torch.nn* library). Where *num_ embeddings* is the vocabulary size (length of the dictionary +1) and *embedding_ dim* is the dimension of the word vectors we want to use (hyperparameter).

2.3 Sentiment classification models

In this part we will introduce the different methods that we implemented to classify the data. we will then talk over its implementation and discuss its performance.

2.3.1 Method 1 : SVM

Method description :

SVM is a supervised machine learning algorithm that can be used to classify the data in different groups. It performs classification by finding the hyper-plane that differentiate the classes we plotted in n-dimensional space. SVM draws that hyperplane by transforming our data with the help of mathematical functions called "Kernels". There is many types of kernels such as linear, sigmoid, RBF, non-linear... In the present project we classify the data in only two groups : positive and negative. Then, a linear classifier is needed.

Method implementation :

To implement this method, we used scikit-learn library. we detailed the method implementation in the Notebook.

2.3.2 Method 2 : Naive Bayes

Method description :

Naïve Bayes classifier is a simple "probabilistic classifier" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.

Method implementation :

To implement this method, we used scikit-learn library. we detailed the method implementation in the Notebook.

2.3.3 Method 3 : CNN

Method description :

Convolutional Neural Networks CNNs can be seen as a special category of Deep Neural Networks. Traditionally, CNNs are used to analyse images. Recently, the idea of using it to classify text has introduced by Yoon Kim, in the paper Convolutional Neural Networks for Sentence Classification [3]. The intuition here is that the appearance of certain bi-grams, tri-grams and n-grams within the review will be a good indication of the final sentiment.

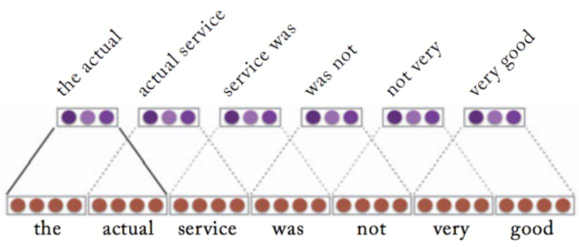

Figure 1: Example of a sentence convolution with k=2 and dimensional output l=3.

Method implementation :

In this part, we designed a full convolution neural network classifier that we called **CNN**. The architecture of the designed CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers consist of a series of convolutional layers followed by pooling layers (local) and fully connected layers. After the convolution layer we applied a *Relu* function and after a fully connected layer we aplied *tanh* function. In the code, we defined the parameters of the CNN, using :

- *X* is the input of the model represented by a sequence of word. we apply to the input an embedding layer with the size of *[vocab_ size, embedding_ dim]*.
- *kernel_ list* defines the number of convolutions by len(kernel_ list) and the size of each convolution kernel.
- *feature_ size* define the feature space.
- *pool_ choice* identify the type of pooling we apply to the output of convolution layers. It can be *max* or *mean*.
- *output* contain the probability of the classification task. If we obtain *output* > 0.5 the review is considering positive, otherwise is negative.

3 Numercial results

In this part, we will evaluate the performance of the classification models we constructed using the three methods : SVM, Naive Bayes and CNN. To do that we can use metrics such as the confusion matrix and the accuracy.

For this three methods, we will evaluate the model by considering different n-gram.

For the SVM and NB methods, we used two different types of data set vectorization : Bag-of-words (*CountVectorizer*) and Tf-IDF (*TfidfVectorizer*). For the two functions, we precised 3 hyperparameters :

- *stop_ words* : it takes the list of stop words to delete from the data set.
- *ngram_ range* : it takes a tuple that contain the lower and upper boundary of the range of n-values for different word n-grams or to be extracted from data set sentences.
- *max_features* : it takes an integer value indicating the maximum number of feature to choose ordered by term frequency across the corpus.

As we said in the second section, for sentiment analysis and especialy for the negative reviews, removing some of stop words has a bad impact on the sentiment classification performance. So for that reason, we constructed a list of custom stop words (custom_ stop_ words =["but","under","no", "not"]). And we evaluate the classification method by deleting all english stop words with the except of custom stop words and we compared the results to the cases where we delete all the english stop words and where we not delete stop words.

3.1 Method 1 : SVM

3.1.1 Bag-of-words

Table 1 : Results without stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	77	76.8	76.4
5600	77.3	77.	77.2
8400	77.6	77.	77.
11200	77.6	76.6	76.8
14000	77.6	77.3	76.9

Table 2 : Results with only custom stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	77.7	77.5	77.2
5600	78.2	78.	77.6
8400	78.5	77.9	77.5
11200	78.5	77.8	77.6
14000	78.5	78.	77.6

Table 3 : Results with stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	77.9	78.286	78.1
5600	78.3	78.250	78.
8400	78.2	79.357	78.9
11200	78.2	79.036	78.7
14000	78.2	79.429	79.

Now, we visualize the confusion matrix for the best model : unigram + bigram, max feature value correspond to the number of features and with stop words.

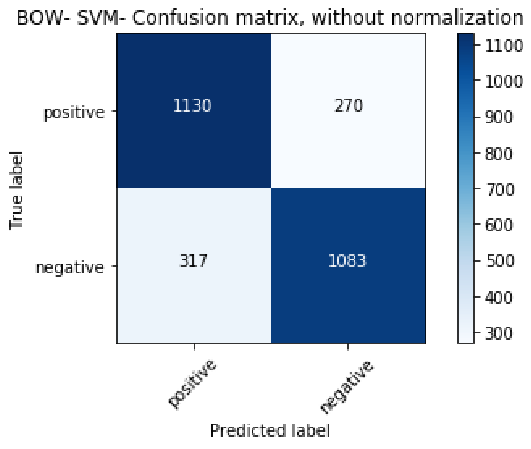

Figure 2: Cofusion matrix.

3.1.2 TF-IDF

Table 4 : Results without stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	78.	77.3	77.1
5600	77.9	78.1	78.3
8400	77.9	77.8	77.8
11200	77.8	78.2	78.3
14000	77.8	78.1	78.4

Table 5 : Results with only custom stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	78.3	77.9	77.7
5600	78.7	78.8	79.1
8400	79.	79.2	79.
11200	78.9	79.1	79.2
14000	78.9	79.4	79.1

Table 6 : Results with stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	79.1	78.7	78.9
5600	79.4	80.2	79.9
8400	79.3	80.2	80.5
11200	79.3	80.3	81.
14000	79.3	80.4	80.8

After getting the results that the highest accuracy value is generated from the ngram with stop words model (TF-IDF and Bow vectorization), then an experiment is conducted to test the accuracy of the bigram and trigram with stop words models to see whether there will be an increase in accuracy. Here we visualize the evolution of test accuracy in terms of number of features (using the values obtained in Table 6), with different ngram range.

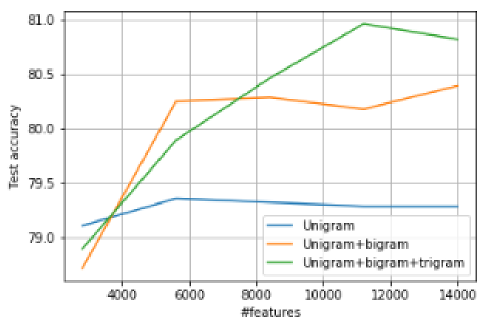

Figure 3: Test accuracy in terms of number of features (*max_features*, using TF-IDF vectorization)

Now, we visualize the confusion matrix for the best model : unigram + bigram + trigram, max feature value equal to 11200 and with stop words (see Table 6).

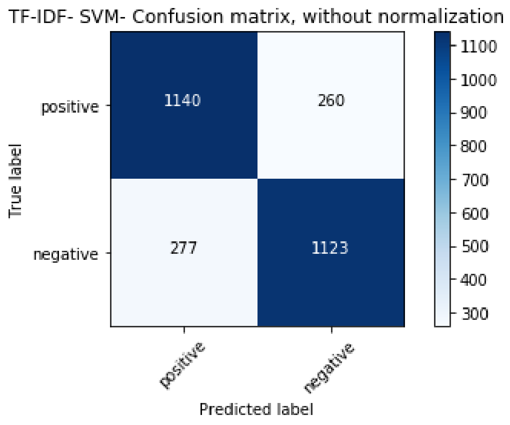

Figure 4: Cofusion matrix.

3.2 Method 2 : Multinomial Naive Bayes

3.2.1 Bag-of-words

Table 7 : Results without stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	76.7	77.2	76.9
5600	77.4	77.1	77.
8400	77.7	77.4	77.1
11200	77.7	77.6	77.2
14000	77.7	77.3	77.3

Table 8 : Results with only custom stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	77.1	77.5	77.536
5600	77.9	77.8	77.571
8400	78.	77.5	77.857
11200	78.	77.9	77.750
14000	78.	78.2	77.821

Table 9 : Results with stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	78.750	78.4	78.393
5600	78.750	79.1	78.714
8400	79.393	79.1	79.107
11200	79.429	79.2	79.214
14000	79.429	79.4	79.286

We visualize the confusion matrix for the best model : unigram + bigram + trigram, max feature value equal to 11200 and with stop words.

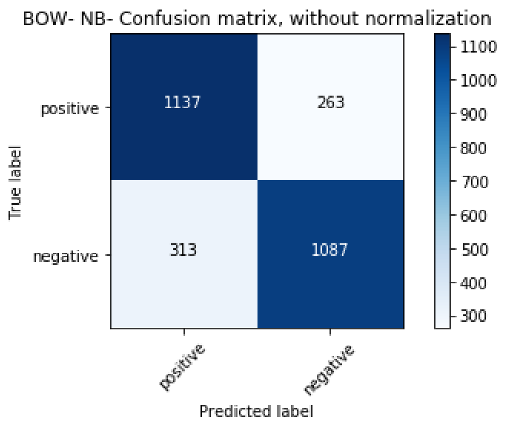

Figure 5: Cofusion matrix.

3.2.2 TF-IDF

Table 10 : Results without stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	76.9	77.2	77.107
5600	77.2	77.9	78.071
8400	77.1	78.2	78.000
11200	77.1	78.1	77.786
14000	77.1	78.1	77.714

Table 11 : Results with only custom stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	77.2	77.5	77.357
5600	77.4	78.1	78.321
8400	77.5	78.4	78.321
11200	77.5	78.3	78.321
14000	77.5	78.2	78.143

Table 12 : Results with stop words

	Test accuracy (%)		
No. of features	1-gram	1-gram + 2-gram	1-gram + 2-gram + 3-gram
2800	78.3	78.8	78.536
5600	78.7	79.5	79.179
8400	78.9	79.5	79.607
11200	79.1	79.3	79.964
14000	79.1	79.3	79.750

Here we visualize the evolution of test accuracy in terms of number of features (using the values obtained in Table 9), with different ngram range.

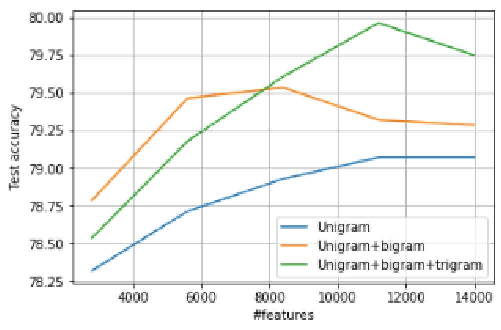

Figure 6: Test accuracy in terms of number of features (*max_features*), using TF-IDF vectorization

We visualize the confusion matrix for the best model : unigram, max feature value equal to 11200 and with stop words.

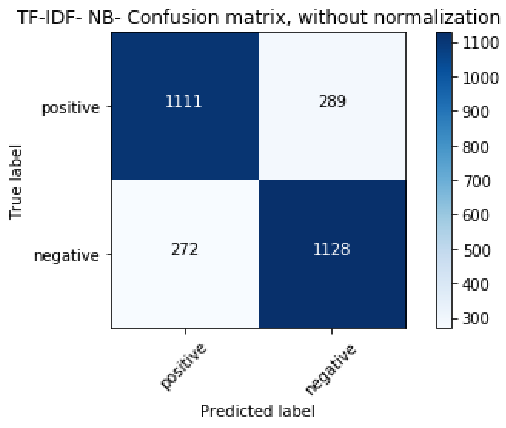

Figure 7: Cofusion matrix.

3.3 Discussion : SVM & NB

- As it was been expected, when we combine BOW with TF-IDF, we clairly see from the tables above that Tf-IDF vectorization ameliorates the test accuracy. For example, we can notice that the test accuracy value improves by +1.8 when using tf-idf vectorization when the max_features = 14000 (see table 3 & table 6).
- Based on the results in the above tables, on the three n-grams models with the conditions mentioned, namely with stop words, without stop words, and with custom stop words, the highest accuracy was generated by models with stop words. For example, in the case of TF-IDF vectorization with SVM method, we achieve an accuracy value of 81.% with the number feature 11200 (see Table 6). In fact, it is right, that using only custom stop words model gives better results than without all stop words. But using all stop words still gives better results, maybe this issue due that we didn't consider all custom stop words.
- When visualizing the evolution of test accuracy in terms of number of features with stop words model (see Figure 2 and Figure 5), we notice that the unigram and bigram with stop words models produce

the best accuracy values depending on the number of features. In fact, when we keep many features, approximately more than 8000 features, trigram model gives better result, otherwise bigram is better. This results are similar for the two SVM methods when using TF-IDF vectorization.

- We notice that SVM method achieve a better accuracy score comparing to Naive Bayes. Actually, the best score obtained by SVM is equal to 81.% (Tf-IDF vectorization) (see Table 6) and the best one achieved by NB is of value 79.964% (see Table 12) (Tf-IDF vectorization). However, Naive Bayes is faster.

- For each method and for each type of vectorization, we visualize the confusion matrix of the model by using hyperparameters that give the best test accuracy (see Figure 2 & 4 & 5 & 7).
  Out of 2800 review, the best classifier (SVM with TF-IDF vectorization) predicted true "positive" 1140 times, and true "negative" 1123 times (see Figure 4).

3.4 Method 3 : CNN

3.4.1 CNN with one convolution layer

<u>Architecture :</u>

- One embedding layer.
- One convolution layer.
- One pooling layer.
- Output layer.

<u>**Test 1 : kernel size 2, feature size 5, pooling choice "max"**</u>

In the following graphs we visualize respectively the accuracy (Figure 8) and the loss (Figure 9) obtained by training the above CNN model.

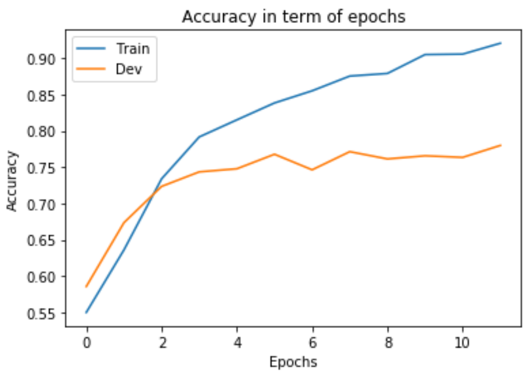

Figure 8: Accuracy on the training and the dev data during the learning step.

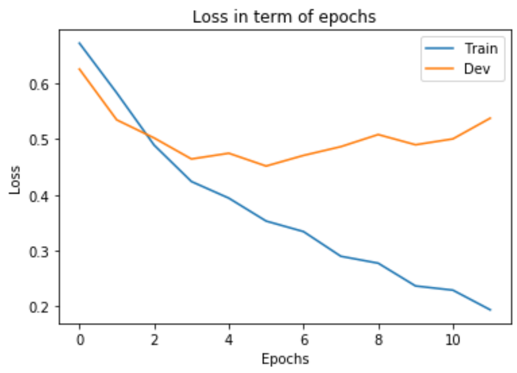

Figure 9: Loss on the training and the dev data during the learning step.

Using this method we obtained a test accuracy equal to 78.015%.

<u>**Test 2 : kernel size 2, feature size 5, pooling choice "mean"**</u>

In the following graphs we visualize respectively the accuracy (Figure 10) and the loss (Figure 11) obtained by training the above CNN model.

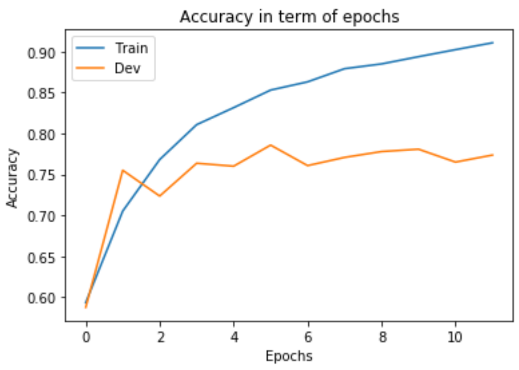

Figure 10: Accuracy on the training and the dev data during the learning step.

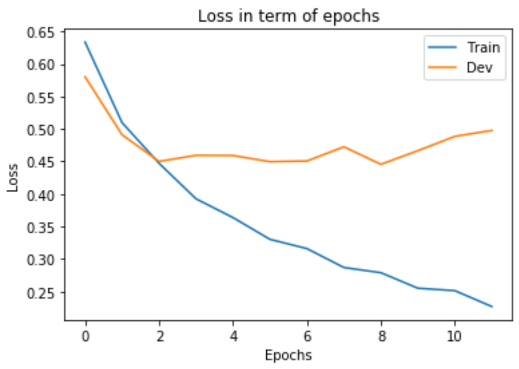

Figure 11: Loss on the training and the dev data during the learning step.

Using this method we obtained a test accuracy equal to 80.790%.

<u>Results and interpretation :</u>

When using mean pooling instead of max pooling, the model start overfitting from the epoch=3 (see figure 5 & 6). However, we obtained a test accuracy of 80.790 % which is better than the accuracy obtained using pooling "max".
So we decided to continue training our model by using mean pooling (hyperparameter).

Test 3 : kernel size 1, feature size 5, pooling choice "mean"

In the following graph we visualize the accuracy on dev and train data sets (Figure 12) obtained by training the above CNN model.

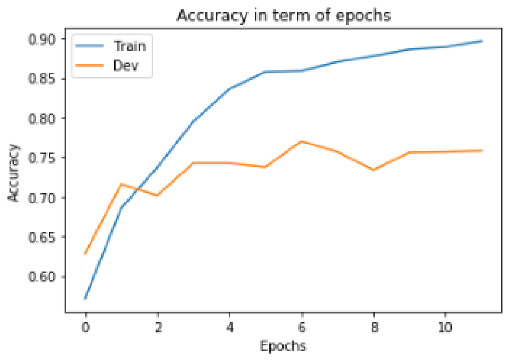

Figure 12: Accuracy on the training and the dev data during the learning step.

Using this method we obtained a test accuracy equal to 75.952%.

3.4.2 CNN with 2 convolution layers

Architecture :

- One embedding layer.
- Two convolution layers.
- Two local pooling layers.
- Output layer.

Test 1: two kernels with size 2 and 3, feature size 5, pooling choice "mean"

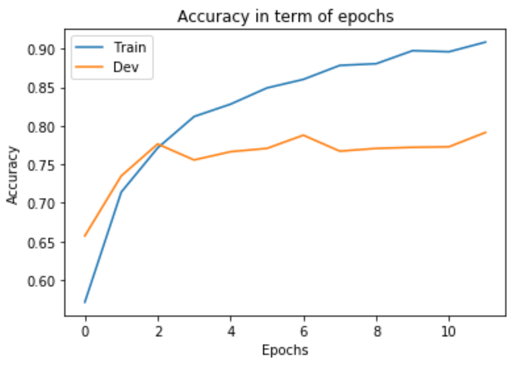

Figure 13: Accuracy on the training and the dev data during the learning step.

In the above graph we visualize the train and dev accuracy in terms of epochs. We notice that the model start overfitting from the epoch = 3. The test accuracy obtained is 81.111%. Then, a CNN with 2 layers give a better test accuracy.

Test 2: two kernels with size 1 and 2, feature size 5, pooling choice "mean"

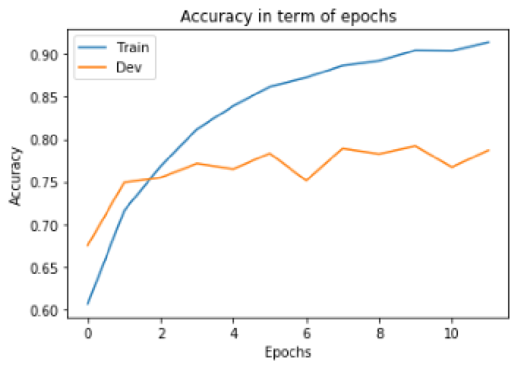

Figure 14: Accuracy on the training and the dev data during the learning step.

In the above graph we visualize the train and dev accuracy in terms of epochs. We notice that the model start overfitting from the epoch = 3. The test accuracy obtained is 80.238%. Then, a CNN with 2 layers give a better test accuracy.

3.4.3 CNN with 3 convolution layers

Architecture :

- One embedding layer.
- Three convolution layers.
- Three local pooling layers.
- Output layer.

Test : three kernels with size 1, 2 and 3, feature size 5, pooling choice "mean"

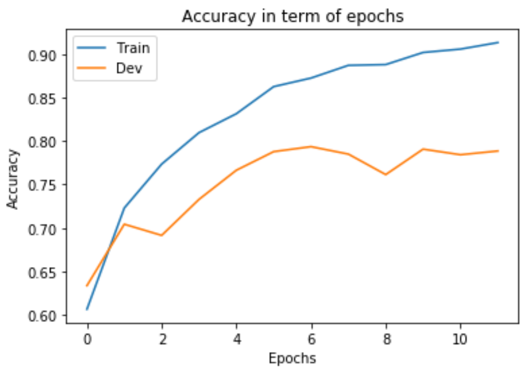

Figure 15: Accuracy on the training and the dev data during the learning step.

Using this method we obtained a test accuracy equal to 80.476%.

3.5 Discussion : CNN

The final result of this project is to find out which classifier model produces better accuracy in sentiment analysis. The classification results obtained in the previous experiments shows that the best accuracy produces by our designed CNN classifier model is 81.111%. Whereas the best accuracy generated by the other classifiers model (SVM & NB) is 81.000%. So, even though the difference is not that big, we can note that the classifier model with Convolutional Neural Network algorithm can provide better accuracy results compared to the Naïve Bayes classifier model in sentiment analysis.

4 Conclusion

The sentiment analysis carried out in this project uses English-language tweets collected from Twitter using the IMDB data set. The entire sentiment analysis process begins from the data preparation process, data vectorization with different methods such as BOW, TF-IDF and embedding, and the use of Convolutional Neural Network, Naïve Bayes and SVM Classifier algorithms is done using the Python programming language.
The use of this methods has been successfully carried out in this project. The best test accuracy result was generated by the CNN classifier. In a futur work, several things for further improvement and development can be done :

- Using more data set so that the classifier model can provide better accuracy in sentiment classification.
- Testing the designed models (SVM & NB & CNN) with more different choices hyperparameters, which the choice has a huge impact in the performance of the model. For that, We can use a good method to tune these hyperparameters such as Grid search method.
- Comparing with other deep learning or machine learning algorithms. We can for example, use reccurent neural network such as LSTM.

References

[1] `https://thesai.org/Downloads/Volume10No5/Paper_11-Comparison_of_Accuracy_between_Convolutional_Neural_Networks.pdf`.

[2] `https://medium.com/@vasista/sentiment-analysis-using-svm-338d418e3ff1`.

[3] `https://www.aclweb.org/anthology/D14-1181.pdf`.