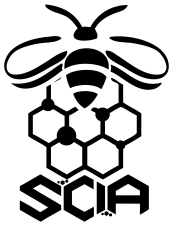
NLP Sentiment Detection Project

— Épita —

SCIA



Kevin Guillet Maxime Leherle Corentin Bunel

October 2021

Contents

Chapter 1

Introduction

1.1 Stakes

This project aims to build a NLP Sentiment Detection AI model for pedagogical purposes. The final result is a benchmark on multiple resolution methods with specific metrics to maximize. All of the result are related to IMDB dataset.

We are going to detail our results with the three methods we have used:

- Naive Bayes model
- Logistic Regression model
- Fast Text Classification model

For each of these models we will first explain how we set up the model, then we will test different optimizations. Each optimization will be tested alone to see if it has a significant effect on the results. Then we will couple the optimizations in order to get the best model. And finally we will finish by analyzing our results in order to understand the qualities and the limits of our model.

1.2 Source code location and explanation

The source code of the project is hosted by Github at this public location:
github of cloud441: NLP project
To re-run the Notebook, please follow this tutorial:

```
# If you are on Linux distro:
virtualenv NLP_env
source NLP_env/bin/activate # select the specific activate according to your shell.
# else, create a Python virtual environment with your IDE, and go inside.

# Install all python modules:
pip install -r requirements.txt


# Run Jupyter Notebook:
# If you are on Linux distro:
jupyter notebook & # the '&' run the command in background to let you use the console.
```

Be careful you can have one hour to run each notebook so if you want to run them again make sure you have time.

The architecture of the code is as follows:

- The 'doc/' directory contains the final benchmark report.
- The 'NaiveBayes/' directory contains notebook of the naive bayes model.
- The 'LogisticRegression/' directory contains notebook of the logistic regression model.
- The 'FasText/' directory contains notebook of models from FastText Text Classification module.

Chapter 2

Modèles & apprentissage

2.1 Naive Bayes model

2.1.1 First model, without pre-processing

The way of thinking Naive Bayes modelization is to understand that each word in a labelled sentence in dataset is present multiple times in other sentences (in our IMDB dataset, inputs are not exactly a sentence but a paragraph, we talk about sentences to simplify explanation). But if all occurrences are in different sentences, there are labelled with different labels. In this case, we can compute the probability of each label according to a word. The Naive Bayes model will compute this probability with counting occurrences of the word in each class. The result is a dictionary that associates a tuple label/word with a probability to occurs. To check the final accuracy of our model, we need to compute if a sentence is composed by a greater number of positive labels than negative, in this case the sentence is predicted as positive. Finally, we just compare our prediction to the ground truth label.

So, our first model is without pre-processing. The classification follows this blueprint:

- each sentences of the IMDB dataset is split into tokens with the python module NLTK tokenizer.
- Pandas Dataframes are filled with train dataset et test dataset to later use of optimized Dataframe tools.
- we factorize the dataframe to extract the dataset vocabulary from train dataset.
- we train our Bayes Model with train dataset and vocabulary to get each loglikelihood value for each label/word tuple.
- As described above, the sentence label is selected by computing each class is the most represented in sentence.

2.1.2 Clean extend optimization

There are some token in IMDB dataset that could be considered as useless to converge the model. So we will filter some of the stop words like: "the", "a", "an", "this", etc ...

The results are:

- train accuracy on words: 0.581
- train accuracy: 0.758

- test accuracy on words: 0.554
- test accuracy: 0.665

2.1.3 Stemming optimization

Another optimization is the stemming that cut verbal form and other derivation in order to simplify the vocabulary. And with this we get the following results:

- train accuracy on words: 0.561
- train accuracy: 0.758

- test accuracy on words: 0.544
- test accuracy: 0.707

2.1.4 Lemming optimization

Lemming is the same idea as stemming but with a more powerful way to detect the root or the base of the words. The objective is to cut the word to the lemma or the minimal form of word.

- train accuracy on words: 0.564
- train accuracy: 0.777

- test accuracy on words: 0.545
- test accuracy: 0.713

2.1.5 Summary

During this model analysis, we keep informations about sentences accuracy but also on words accuracy to compare if there is a logic relation between these two metrics. As you can see above, There is no logic between them. Have a good accuracy on word classification is not a clue that this strategy will give a better accuracy on IMDB classification.

Let's compare our pre-processing strategies. Stop word filtering looks to be a way to lose needed informations for Bayes Naive model to converge on sentences. So we will consider that this strategy is bad.

Stemming doesn't look to be efficient because we lose a little bit of accuracy on test dataset. Lemmatization seems to be a better fit with our model. The result are not so much better but It doesn't mean that the strategy is useless with IMDB dataset. This

conclusion could be the same for stemming strategy. But why do we think that these two strategies could be an interesting solution?

The reason is that our Naive Bayes model is not so smart. To classify a sentence by analysing each word separately is a bad approach in NLP problem. Context analysis is needed and have the possibility to reduce complicated word to lemma or reduce stemming fit well this resolution approach.

2.2 Logistic Regression model

2.2.1 First model, Features only

For this model we will need to create features from our text to use as input to our logistic regression. We will use the following features:

- 1 if "no" appear in the doc, 0 otherwise.
- The count of first and second pronouns in the document.
- The count of first and second pronouns in the document.
- The count of "." => number of classic sentences.
- 1 if "!" is in the document, 0 otherwise.
- log(word count in the document).
- Number of words in the document which are in the positive lexicon.
- Number of words in the document which are in the negative lexicon
- Number of words in the document wich are in the lexicon but netral

So we have used the basic features proposed then we have added 2 new features and we are now going to explain our choice. For the feature sentences count we noticed that often the positive criticism contained shorter sentences, indeed they are less detailed than the sentences with a negative connotation so that's why we want to add this feature. And then for neutral words we decided to choose this because we already use the number of words but here we use the number of interesting words that add meaning to the text.

We are talking about a lexicon, this lexicon is a text file containing words with an associated score and it is with this lexicon that we will be able to calculate the last 3 features. We use the following rule : the words between -1,5 and 1,5 will be neutral, the words has 1,5 and more will be positive and those has -1,5 and less will be negative.

With this basic model we have the following scores:

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.6965906303654648
        - recall :  0.6890137883627835
        - fbeta_score :  0.6816


For labbel 1 :
        - number of entry :  12500
        - precision :  0.6883076200172292
        - recall :  0.6956349677470418
        - fbeta_score :  0.70312
```

First of all we can see that we have 12500 elements of each label.

So we can see that the results are quite correct. We have a precision and a recall between 0,68 and 0,70 on each label. The fbeta score, which is the weighted harmonic mean of precision and recall, is of 0,68 for label 0 and 0.70 for label 1.

2.2.2 Clean optimization

In the text of the elements, there are still some formatting characters in fact there are still "()", ";" and so on. So we will define a list of words and formatting characters that we will remove from the text. Moreover we will also put spaces before and after the "!" so that they are considered as words.

And with this method we have a calculation time that is certainly longer but still remains in the order of a minute, so it is still very low. And we obtain the following results:

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.6965906303654648
        - recall :  0.6890137883627835
        - fbeta_score :  0.6816


For labbel 1 :
        - number of entry :  12500
        - precision :  0.6883076200172292
        - recall :  0.6956349677470418
        - fbeta_score :  0.70312
```

We can see with the scores that the optimization do not change the results, in fact it only modifies the features a little and it is not enough to be able to modify the prediction of an element, so we will not keep this optimization.

2.2.3 Stemming opimization

We will now try to use stemming on our text to see if it changes our results. And with this we get the following results:

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.6702224427354668
        - recall :  0.6591306469320538
        - fbeta_score :  0.6484


For labbel 1 :
        - number of entry :  12500
        - precision :  0.6594871000232432
        - recall :  0.6700515605935372
        - fbeta_score :  0.68096
```

We can see that the results obtained are worse than without optimization which is strange because normally we should keep only the meaning of the words and therefore be able to compare them better. And we finally understood why our results were not good, indeed we compare words that are stemmed with a dictionary of words that are not stemmed and therefore the match can not be done. So we modified our dictionary by applying the stemming function to its words and we get the following results:

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.692652329749104
        - recall :  0.6863900548918308
        - fbeta_score :  0.68024


For labbel 1 :
        - number of entry :  12500
        - precision :  0.6858692235146181
        - recall :  0.6919600380589915
        - fbeta_score :  0.69816
```

We can see that the results are of the same order of validation but as the calculus time is not significantly longer we will keep this optimization.

2.2.4 Lemming optimization

We will try to see with this method what we can obtain.

We have therefore applied the lemming method to all our texts and also to our dictionary in order not to have the same problem as when we tested the stemming method. And we obtain the following results:

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.5998505976095617
        - recall :  0.46931618936294567
        - fbeta_score :  0.38544


For labbel 1 :
        - number of entry :  12500
        - precision :  0.5472654408297972
        - recall :  0.630242975430976
        - fbeta_score :  0.74288
```

We can see that the results of the lemming are not satisfactory in fact we think that this comes from the fact that the words are too modified and that by this fact some feature as the calculation of the number of pronouns of the 2nd person become are not well calculated and therefore this causes a loss in the results. And it also takes much more time to calculate and we arrive here at a calculation time of about 1 hour which is too much compared to times of the order of a minute with the previous methods.

2.2.5 Regularization

We also tested all the regularizations available in our model, so we did the tests with:

- L1 regularization
- L2 regularization
- Elasticnet regularization
- and without regularization

And we have obtained the following results:

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.6965906303654648
        - recall :  0.6890137883627835
        - fbeta_score :  0.6816


For labbel 1 :
        - number of entry :  12500
        - precision :  0.6883076200172292
        - recall :  0.6956349677470418
        - fbeta_score :  0.70312
```

L1 regularization

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.6965906303654648
        - recall :  0.6890137883627835
        - fbeta_score :  0.6816


For labbel 1 :
        - number of entry :  12500
        - precision :  0.6883076200172292
        - recall :  0.6956349677470418
        - fbeta_score :  0.70312
```

L2 regularization

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.6965906303654648
        - recall :  0.6890137883627835
        - fbeta_score :  0.6816

For labbel 1 :
        - number of entry :  12500
        - precision :  0.6883076200172292
        - recall :  0.6956349677470418
        - fbeta_score :  0.70312
```

Elasticnet regularization

```
For labbel 0 :
        - number of entry :  12500
        - precision :  0.6965906303654648
        - recall :  0.6890137883627835
        - fbeta_score :  0.6816

For labbel 1 :
        - number of entry :  12500
        - precision :  0.6883076200172292
        - recall :  0.6956349677470418
        - fbeta_score :  0.70312
```

without regularization

So we can see that the regularization doesn't change the result, so we'll stay on the proposed default regularization function.

2.2.6 Summary

With these examples and others that we have studied, we can easily see some problems that our classifier may have:

- It does not detect irony in sentences, i.e. if a person uses irony there is a high chance that his text will be classified in the other category.
- It also doesn't detect false compliments, for example if someone says "This is a great ninja movie for kids" it will just keep the positive side when it can mean that it is an interesting movie when you have a childish vision of the story.
- When critics use certain expressions, it can also be a misclassification, often expressions with several meanings that are not necessarily well understood.
- When the text is too descriptive, it takes over the possible short passages with the emphasis on the feeling and so these cases are at the border of the two classes and so are sometimes misclassified.

These are the major problems, there are surely other less visible ones that we have not seen yet. But overall we have a result close to 70% which for a method that only takes a few minutes to execute is quite correct from our point of view.

2.3 FastText Text Classification model

2.3.1 First model

To start, we will make a basic model with fasttext to have a point of comparison and to see if the different improvements really bring an upgrade of the model. With this model we have the following results :

```
label '__label__1':

        precision: 0.867
        recall: nan
        F1 score: 1.733

label '__label__0':

        precision: 0.852
        recall: nan
        F1 score: 1.704
```

2.3.2 Clean optimization

we will use the same cleanning optimization as in logistic regression

And we obtain the following results:

```
label '__label__0':

        precision: 0.882
        recall: nan
        F1 score: 1.763

label '__label__1':

        precision: 0.876
        recall: nan
        F1 score: 1.752
```

So we can see that this optimization improves our results and so we will keep this one for now.

2.3.3 Clean extend optimization

We have see that the function to clean upgrade our result but why stop here?

We can add another clean step on the text. This step is to delete stop words. What is stop words? Stop words are the non-discriminating words, like "the", "a", "an", "this"...

We can now test with our new clean function:

```
label '__label__1':

        precision: 0.891
        recall: nan
        F1 score: 1.781

label '__label__0':

        precision: 0.881
        recall: nan
        F1 score: 1.762
```

So with this result we can see that the results are a bit better so we will now use the clean text to expand instead of the clean text classic.

2.3.4 Stemming optimization

As for the logistic regression, once again we will see if the stemming method allows us to improve the results. And with this we get the following results:

```
label '__label__1':

        precision: 0.86
        recall: nan
        F1 score: 1.72

label '__label__0':

        precision: 0.863
        recall: nan
        F1 score: 1.726
```

We have an improvement but it is very weak so we can't say that this optimization is really interesting, indeed it is fast but the improvement is too weak to add complexity to our model.

2.3.5 Lemming optimization

Now with lemming :

```
label '__label__0':

        precision: 0.868
        recall: nan
        F1 score: 1.736

label '__label__1':

        precision: 0.867
        recall: nan
        F1 score: 1.734
```

We can see that this time the improvement is a little more interesting, even if it is very long (about 1 hour on the computer where we did the test). But we will keep it because it is quite long but it allows an improvement and we will see how it couples with the other optimization.

2.3.6 Hyperparameters tunning

We are now going to use a fast text function that allows us to find the best parameters for our model. To do this we need to split the train set, in fact we will take a part for train and another for validation and with this validation part we will be able to find the optimal parameters for our model. We will test this function on the basic model to see if it brings an improvement.

```
label '__label__1':

        precision: 0.882
        recall: nan
        F1 score: 1.764

label '__label__0':

        precision: 0.885
        recall: nan
        F1 score: 1.769
```

Results are better with tunning, and we highlight that optimize f1 result on negative label induces better improvements on positive label. The reason is that we just have two labels and negative label had less wrongly classification.

2.3.7 Final model

We have now tested all the optimizations we want to test. So we're going to take three optimizations and couple them, we're going to use the clean text extend function, we're also going to add the lemming function and finally we're going to take this model and use the fast text function to optimize the parameters of our model. With all this optimization we get our final model which has the following results:

```
label '__label__1':

        precision: 0.897
        recall: nan
        F1 score: 1.795

label '__label__0':

        precision: 0.891
        recall: nan
        F1 score: 1.782
```

2.3.8 Summary

So we arrive with our final model to really satisfactory results for what we wanted we get close to 90% accuracy on the two labels. Even if this model is quite long (about 1h20 to do everything) we find that it is a pretty good time for this result. But we will see examples of misclassified texts to try to understand why they are misclassified and see the weakness of our model.

After analyzing some examples, we understand in fact the problem of our model on the 10% error. Indeed, it does not take into account the context. Indeed, a person can say that a movie is good because it is different from other movies, but if he says that there is a strong chance that this person will give a negative feeling on the other movies he is talking about and so as our classifier does not take into account the context he will think that these negative adjectives are used for the movie. And the second problem of this contextualization can be seen with expressions with negative connotations but used in a positive context for example "with an extremely dark humor", this is not necessarily negative but as the terms used are we arrive at a classification problem.

So to summarize our model has a very good success rate and the big problem that can remain to solve is the context handle but which is much more complicated.

Chapter 3

Benchmark

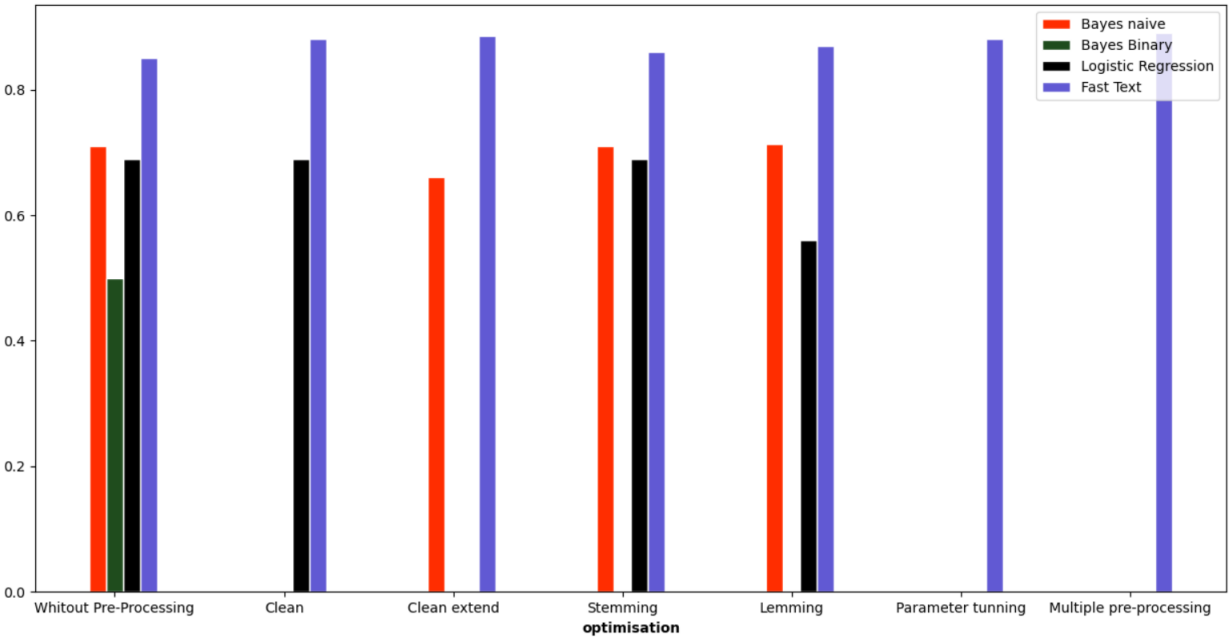
3.1 Metrics & analysis

Our analysis will compare each model with the different optimizations to see the best values of each model. Our stats use different metrics like accuracy, precision, recall and f1 score.

3.2 Benchmark table

	Without Pre-processing	Clean	Clean extend	Stemming	Lemming	Parameter tunning	Multiple pre-processing
Bayes naive	0.71	X	0.66	0.71	0.713	X	X
Bayes binary	0.5	X	X	X	X	X	X
Logistic Regression	0.69	0.69	X	0.69	0.56	X	0.69
Fast Text	0.85	0.88	0.885	0.86	0.87	0.88	0.89

3.3 Benchmark barplot



3.4 Analysis of Results

As you can see above, the model that manages to get the best results is FastText text classifier. But the Bayes Naive model gives acceptable results and this is better than logistic regression.
Moreover, our different pre-processing strategy doesn't look to be as efficient on all models. The logistic regression is pretty bad and it is independent to the strategy. The Bayes model results change a bit but we think that this model is not enough smart to understand context. And the aim of stemmming and lemmatization is to simplify sentence in order to the model could recognize basic structure. Although, the FastText model is able to gain some pourcentage with specific pre-prcessing startegy or with parameter tuning. And we get the best result with mixing some of our strategy: clean extend stop word, lemmatization and model parameter tuning.

Chapter 4

Conclusion

To conclude our project, we managed to get almost 0.9 of accuracy on each label. This result is very good according to us. But we can ask ourselves how to improve this result. Indeed, our score of 0.9 is good, but it still means that in 10% of the cases our model does not predict well the text label. Depending on the problem 10% of error can be too much so we really need to understand where this problem comes from.

In view of the different analysis, we have done during the different parts we have a good idea of what is the problem. Globally the problem of our model is the context, indeed taking into account the context is very important when working on film reviews because irony or false compliments are quite present in these texts. Apart from that we also need to know if a sentence is for the current movie or not, for example we can have a negative sentence about another movie (like the previous one of the current one) to put forward the positive side of this new movie. But all this must be done through the understanding of the context.

So to finish this model, we should add some understanding of the context even though this is a much more complex part and requires much more research.