Big Data Algorithms, Techniques and Platforms
Project report

Soufiane Hadji, Josias Kayo Kouokam, Carmelo Micciche January 24, 2019

The goal of this project is to analyze massive data and highlight the advantages (and drawbacks) of big data approaches, in each step of the the process. We decided to work on a Kaggle classification challenge and compare scikit-learn with pyspark (Apache Spark).

1 Presentation of the objective

1.1 Subject

The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

The study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion) provided some solutions, but so far publicly available models don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content).

1.2 Goal

The goal is to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate better than Perspective's current models, using a dataset of comments from Wikipedia's talk page edits (68Mo, 160k comments). Improvements to the current model will hopefully help online discussion become more productive and respectful.

The data contains for each sample (i.e. each comment) the comment id and the comment text. For the training set, six label columns describing types of toxicity (toxic, severe toxic, obscene, threat, insult, identity hate) are given and contain 0 or 1. This is therefore a binary classification task, repeated six times (one for each type of toxicity).

1.3 Processing steps

The three main steps are data cleaning (or preprocessing), data vectorization (or data quantification) to replace words with their number of occurence, or more precisely by a list of features which measure their relative frequency of appearance based on a determined list of group of words. Once the design matrix built (matrix containing the features), we do a classification using a logistic regression.
The data cleaning is usually the trickiest and time-consuming step of the process, but it is necessary to get good performance for the prediction task.

2 Data processing

Processing has been done using different cluster settings (2 and 8 instances of 4Go) and we controlled the number of cores (1 and 4) on a MacbookPro with 2,5 (GHz), RAM 8Go.

2.1 Data cleaning

The cleaning is composed of different steps to make sure that the data we feed to the vectorizer is usable. We lower the text and remove *stopwords*. Stop words are words which are filtered out, they refer to the most common words in a language, there is no single universal list of stop words, we decided to use the nltk library for this task.

We chose a logarithmic xscale for the plot of cleaning time w.r.t sample size. Scalability results expected are found :

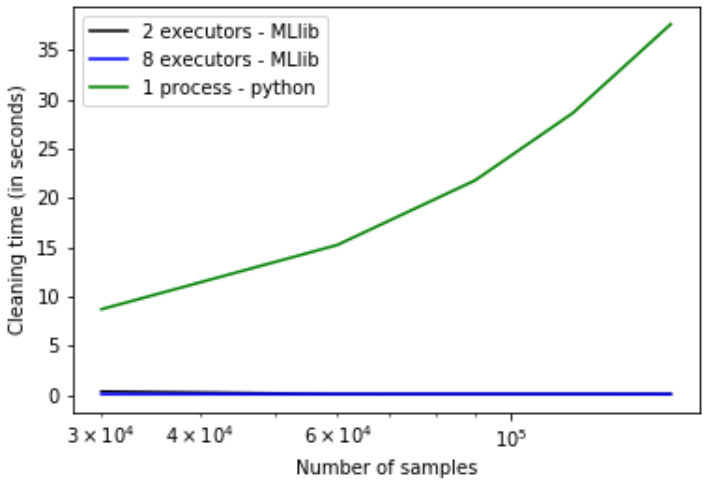

Figure 1 : Data cleaning. Evolution of cleaning time for different versions of Spark (2, 8 executors) and scikit-learn (1 process)

We can conclude that a Spark architecture is more efficient than an integrated architecture for this cleaning step.

2.2 Data vectorization

We will study here the process of vectorization of the data. From the lowered and cleaned comments, we tokenize the data (transform sentences into a list of words) and use the HashingTF then IDF functions to represent our words in a quantitative object. In the hashing function, each word is given a unique index, the number of values possible in this attribution is a parameter n_hash. Therefore words are mapped to a predetermined number of possible codes. The IDF function measures the importance of the word in the corpus of comments and gives more important to infrequent words, because they are better discriminators between comments.

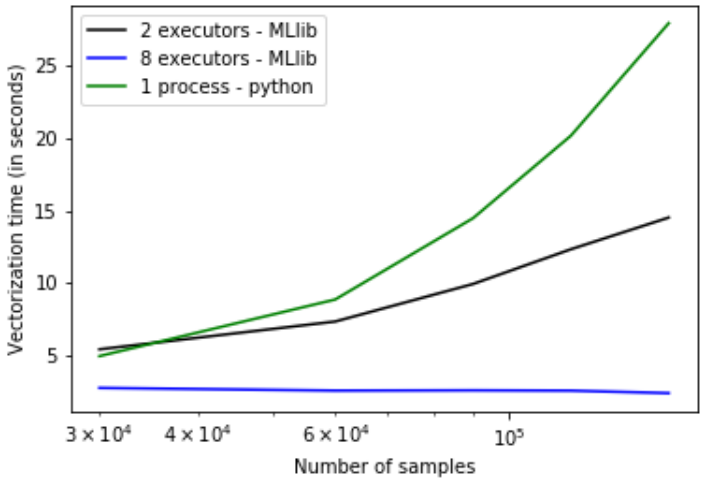

Figure 2 : Data vectorization. Evolution of vectorization time for different versions of Spark (2, 8 executors) and scikit-learn (1 process)

The Spark architecture seems to be very efficient compared to the Python one as well for the vectorization task.

2.3 Learning model

2.3.1 Time performance

We implemented a logistic regression using MLlib (Spark) and sklearn (Python) and compared execution time. Actually we implemented 6 logistic regressions, one for each comment type to detect (toxic, severe toxic, obscene, threat, insult, identity hate). In terms of time performance, here are the performances :

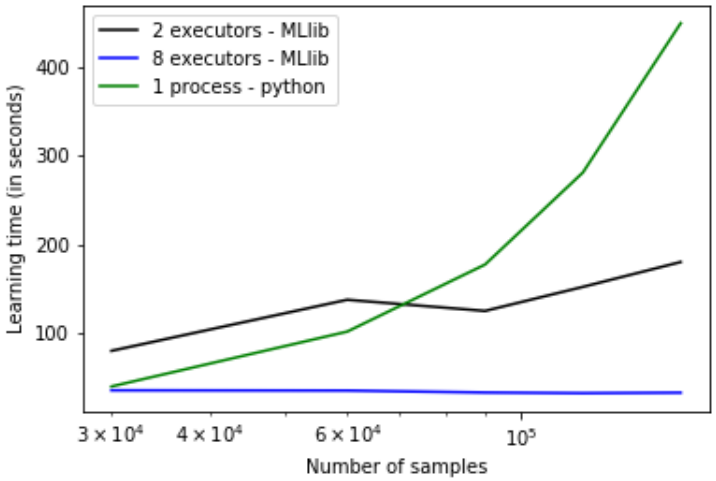

Figure 3 : Classification task. Evolution of learning time for different versions of Spark (2, 8 executors) and scikit-learn (1 process)

It is interesting to see that for a small sample size and only one process, Python's sklearn performs better than a 2-instance Spark architecture. From a certain number of samples (around 70000 samples here), scalability issues appear for Python and Spark becomes a clearly better solution. The Spark architectures with 8 executors is very efficient overall.

2.3.2 Influence of n_hash, number of words in the vocabulary

We plot the learning time and test error rate evolutions w.r.t sample size, for different values of *n_hash* : 500, 2000 and 10000. For the test error, we considered the classification task of the "toxic" comments.

Spark :

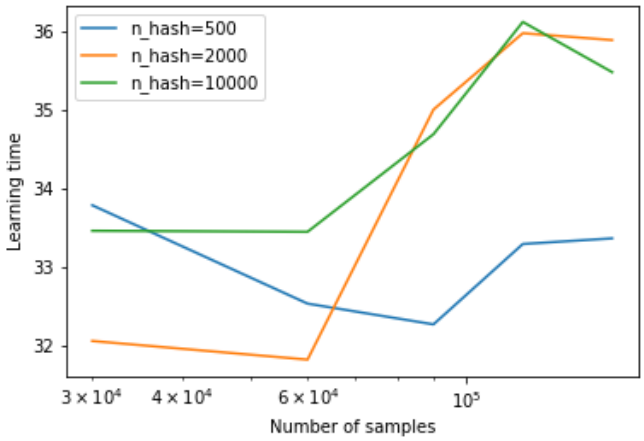

Figure 4 : Evolution of learning time for Spark for different values of the parameter

We find, as expected, that the number of words in the learning time grows with the complexity of the vocabulary (and therefore the *n_hash* parameter).

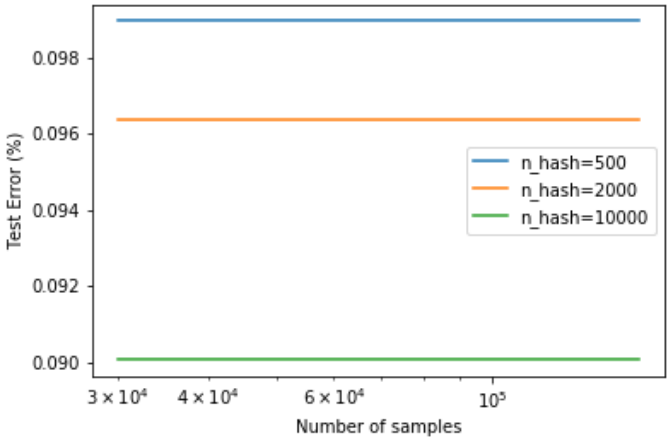

Figure 5 : Evolution of test error rate for Spark for different values of the parameter

The surprising thing here is that it seems like the number of samples does not influence on the performance of the prediction. We would have expected that the test error would decrease when we add more samples to the dataset. Nevertheless, we can see that the *n_hash* parameter has a clear influence on that performance. The bigger the vocabulary, the better the performance. We gain almost 1% of accuracy by using a 10000 hashing parameter rather than a 500 hashing parameter.

Scikit-learn (Python)

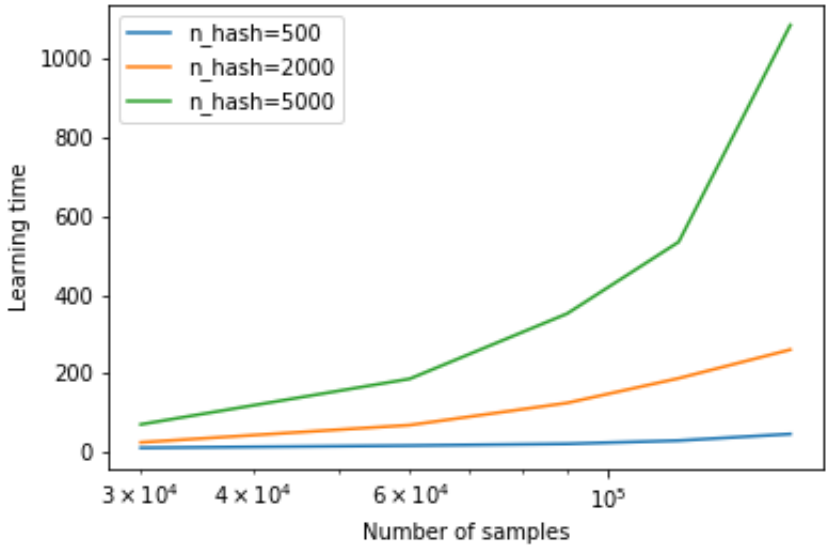

Figure 6 : Evolution of learning time for Scikit-learn for different values of the parameter

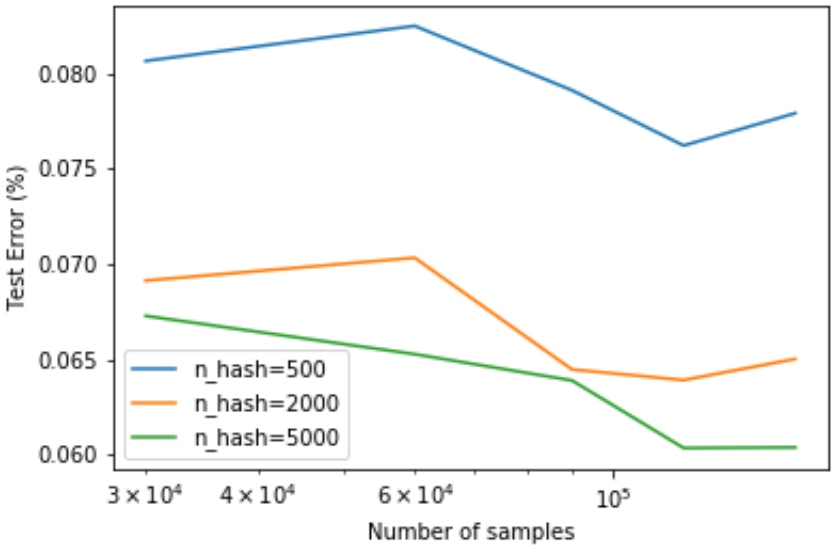

Figure 5 : Evolution of test error rate for Scikit-learn for different values of the parameter

Analysis :

- We have the same trend as with Spark, the bigger the vocabulary, the better the performance, this is natural considering the algorithm will tend towards unicity for word encoding.

- For each value of the parameter, we can see that increasing the number of samples helps reduce the test error, which is something we could expect, and did not see clearly with the Spark approach since the error was almost constant around a value. Obviously with more data, the model is better trained and generalizes better.

- Prediction performance is better with Python with test error values lower than corresponding ones in Spark, with same parameter choice.

3 Conclusions

We implemented a successful algorithm of classification of comment texts, and compared performances with Spark and Scikit-learn. While the cleaning, vectorization and learning time show the efficiency of Spark, the difference with Python on small datasets is slight. Scikit-learn show great quality on prediction tasks as well. Spark should thus be used for massive datasets and can be expected to perform very well in such cases.

References

[1] Spark Apache documentation : running on YARN, feature extraction, stopwords remover, model selection aka hyperparameter tuning
[2] *How to process textual data using TF-IDF in Python*
https ://medium.freecodecamp.org/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3
[3] *Prepare text data in scikit-learn*
https ://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/