COMP47750 - Cloud Computing Project
-
Hadoop/MapReduce implementation for Fake News Detection
-

December 3, 2021

Aleryc SERRANIA
N°21204068

Contents

1 Introduction

This report is the synthesis of the research and implementation of the COMP47780 - Cloud Computing module project.

The aim of this project is to implement the Hadoop/MapReduce model in order to detect articles and sources propagating fake news, a major scourge in our society where misinformation is spreading more and more rapidly because of the massive use of social networks, which leads to some disastrous political, economic and ecological repercussions.

With the 2022 presidential elections in France approaching, articles related to the candidates and their political programmes are proliferating and the risk of spreading disinformation is all the greater as the time devoted to checking sources among the French is reduced.

There are already many prediction models and applications for detecting fake news for English related sources (ClaimBuster, Adverif.ai, Alto Analytics, Grove). From this point of view, I think it is interesting to look at French-speaking sources, especially in this political context of pedidential elections.

The interest in using cloud technologies in this project is multiple:

- on the user side to be able to obtain an analysis and an application usable by all since it only requires an internet connection and a web browser
- on the developer side to take advantage of the speed of computing power, scalability, and reliability provided by cloud platforms to quickly and easily build a reliable application

The MapReduce model allows us to analyze our dataset in a parallel fashion in order to optimize and accelerate its processing when its size is significant. This model is adapted to cloud technologies since we can easily separate the processing on different virtual machines deployed on the cloud.

2 Specifications and requirements

2.1 Specifications

The main specifications are the following:

- Design a supervised learning algorithm to build a binary classification model of articles written in French related to politics and the upcoming presidential elections in order to discern fake news from real news.
- To propose an interface to analyse a dataset in order to build a classification model
- Propose an interface to evaluate this model with measures (recall, precision, accuracy, F1 measure, etc.) and relevant graphs (confusion matrix, words popularity, etc.)
- Propose an interface to test and classify new articles using the previously trained model

2.2 Requirements

The project requirements for its completion are as follows:

- Collect a representative dataset of recent political articles of misinformation from different sources
- Collect a representative dataset of political articles from various reliable sources

- Use the MapReduce model to train the binary classification algorithm within a cloud architecture
- Deploy the application back-end and interface on one or more instances in the cloud

3 Learning Algorithm: The Naive Bayes Classification

There are several approaches to classification algorithms such as Support Vector Machines (SVM) or Neural Networks (CNN and RNN). I decided to focus on the Naive Bayes statistical algorithm since it is the classification method I am most familiar with and it remains relevant for text classification. I also find that adapting it to integrate it into the MapReduce model is an interesting challenge.

3.1 Algorithm explained

The Naive Bayes approach consists of using Bayes' theorem on conditional probabilities:

$$P(h, D) = \frac{P(D|h)P(h)}{P(D)}$$

with:

- $P(h)$: the probability of encountering the hypothesis h in our dataset
- $P(D)$: the probability that a set D exists in our dataset
- $P(h|D)$: the probability that the hypothesis h is true when we know a set of data D
- $P(D|h)$: the probability of obtaining a set D if we know that the hypothesis h is true

In the context of this project, we are interested in finding the probability that an article is fake or real by knowing the words used, i.e. calculating $P(fake|\{word_1, word_2, word_3, ...\})$ and $P(real|\{word_1, word_2, word_3, ...\})$ and comparing them:

- $P(fake|D) > P(real|D) \implies$ article is fake
- $P(real|D) > P(fake|D) \implies$ article is real
- $P(fake|D) = P(real|D) \implies$ we don't know

With a training dataset where articles are labeled, we know the values of $P(fake)$ and $P(real)$:

$$P(fake) = \frac{Number\,of\,fake\,articles}{Number\,of\,articles}$$

$$P(real) = \frac{Number\,of\,real\,articles}{Number\,of\,articles}$$

$P(D)$ is not required as we can just multiply the probabilities $P(fake|D)$ and $P(real|D)$ by $P(D)$ and then check which value is highest.

The main problem comes from the estimation of the value $P(\{word_1, word_2, word_3, ...\}|fake)$ which is too complex to be calculated. This is where we introduce the "naive" part of this algorithm: we make the strong assumption that the probability of a word's appearance is independent of all the other words. Thus we can reduce the calculation of this value like this:

$$P(\{word_1, word_2, word_3, ...\}|fake) = \prod_i P(word_i|fake)$$

with

$$P(word_i|fake) = \frac{Number\,of\,occurences\,of\,word_i}{Number\,of\,fake\,articles}$$

3.2 Laplace smoothing

On remarque rapidement qu'un problème survient lorsqu'un mot apparait dans une classe mais pas dans l'autre, puisqu'on calcule un produit, the likelihood devient 0 directement sans prendre en compte les autres mots.

Pour remédier à ce problème, on peut utiliser une technique de smoothing appelé Laplace Smoothing qui permet de lisser tous les comptes pour enlever les 0. Concrètement la formule est la suivante:

$$P(fake|word_i) = \frac{Number\,of\,occurences\,of\,word_i + 1}{Number\,of\,fake\,articles + |V|}$$

avec $|V|$ la taille du vocabulaire de notre jeu de données.

3.3 Dealing with small probabilities

Puisque le nombre de mots dans un article est élévé (de l'ordre 10^3 ou 10^4), on risque de rencontrer des probabilités extrèmement petite lors du calcul de $\prod_i P(word_i|fake)$.

Par exemple, en comptant 100 mots avec chacun une probabilité $P(word_i|fake) = 2^{-4}$, on se retrouve avec une probabilité totale de 2^{-400}. Stocker cette valeur dans un double n'est donc pas envisageable puisqu'on se retrouvera avec un underflow arithmétique qui donnera une valeur faussée.

Pour remédier à ce problème, on peut calculer le logarihtme de cette probabilité puisque pour une valeur probabilistique comprise dans $[0, 1]$, il renvoit une valeur entre $[-\infty, 0]$ tendant lentement vers l'infini lorsqu'on tend vers des valeurs très petites. Par exemple, $log(2^{-400}) \approx -120.41$ et est bien plus raisonnable pour une machine 64 bits.

En utilisant les propriétés du logarithme sur les produits, on arrive au résultat suivant:

$$log(P(fake|\{word_1, word_2, word_3, ...\})) = log(P(fake) * \prod_i P(word_i|fake))$$

$$log(P(fake|\{word_1, word_2, word_3, ...\})) = log(P(fake)) + \sum_i log(P(word_i|fake))$$

3.4 Advantages and Limitations

L'avantage de cette algorithme est qu'il est assez rapide à implémenter: le principe de base est intuitif et simple à comprendre mais permet d'avoir de bons résultats de classification. It's also suitable for moderate or large data sets with many words and can deal with missing feature (in our case, words that have never appeared during the training phase)

Cependant l'algorithme Naive Bayes suppose l'indépendance des features, ce qui est une hypothèse forte dans l'apparition des mots et qui est violée dans la majorité des cas. Généralement dans n'importe quel texte l'apparition d'un mot dépend fortement du contexte de sa phrase. De plus il ne prend pas en compte la structure d'un

4 Data collection and pre-processing

4.1 Data collection

The data collection was carried out in three stages:

- Finding sources for reliable articles
- Finding sources for fake news articles
- Retrieving and storing the articles for each identified source

In order to simplify the process of categorising articles between fake and real, I based my analysis on the source of the articles: if an article comes from an author or a website previously identified as a fake news source, then it will be categorised as fake news.

Normally, each article would have to be analysed and fact-checked in order to label them precisely, but this process would unfortunately take far too much time for the duration of this project. Of course, the possible biases that this decision making might induce will have to be taken into account when evaluating the model.

In order to identify a source as reliable or not, I used the Decodex, a database maintained and updated by the newspaper *Le Monde* which lists sources of fake news propagation.

For articles from reliable sources, I collected articles from newspapers of different political side:

- Le Monde: centre left
  - https://www.lemonde.fr/politique/
  - https://www.lemonde.fr/election-presidentielle-2022/
- Le Figaro: right wing
  - https://www.lefigaro.fr/elections/presidentielles
  - https://www.lefigaro.fr/politique
- Le Nouvel Observateur: left wing
  - https://www.nouvelobs.com/election-presidentielle-2022

For articles from unreliable sources, I have identified the following:

- Medias Presse Info: far-right site known for its conspiracy theories and false quotes
  - https://www.medias-presse.info/category/politique/
- Riposte Laique: far-right site with islamophobic articles and hate speech
  - https://ripostelaique.com/category/collabos
- Le Gorafi: satirical site that parodies French news

    - https://www.legorafi.fr/category/france/politique
- Franche Info: parody site of humorous articles inspired by reality
    - http://franchetvinfo.fr/catégorie/politique

In total, more than 3500 articles were collected, with more than 1800 articles from unreliable sources, for a total size of 10 MB.

4.2 Data pre-processing

The pre-processing stage of data takes place in three steps: cleaning, stemming and vectorisation.

4.2.1 Cleaning

The cleaning step consists in extracting important words by removing punctuation, words with little semantic value (called stop words), numbers, urls and other symbols related to the web format (emoticons, etc.). To detect and remove URLs, I used the regular expression of gruber on Github.

Correcting or deleting grammatical errors and discarding nonsense words could also be a good idea but it would require a more thorough and complex analysis of the language. In addition, it might also inadvertently discard neologisms used by certain types of media that would be interesting to keep in the analysis.

4.2.2 Stemming

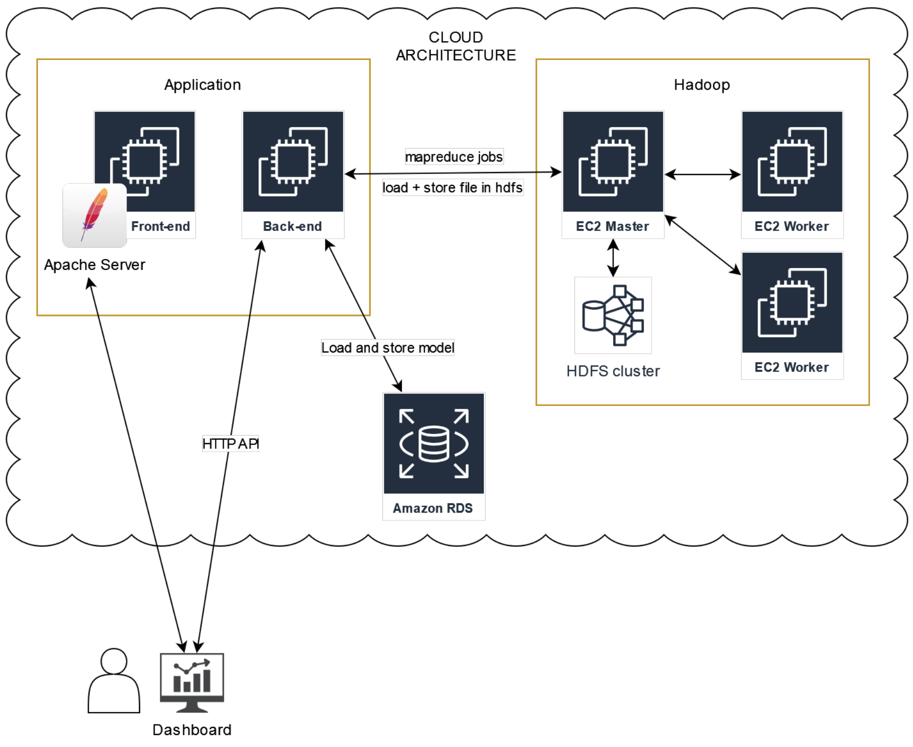
Stemming is the extraction of the root of a word in order to gather in a single token the small grammatical variations of the same word. This principle makes it possible to improve the performance of a text analyzer. Stemming algorithms are language-specific. For French, we can use the French Snowball Stemmer.

4.2.3 Vectorisation

Vectoring is the transformation of a text into a vector of size determined by the vocabulary of the data set. More simply, each possible word is assigned a position in a vector of numbers and this number represents the number of occurrences in the text. Since the number of words used in an article is very small compared to the total vocabulary, the vectors will mostly contain 0s. Therefore, it is preferable to store words in a dictionary whose key are the words and values are the number of occurrences, in order to reduce storage costs.

5 System Architecture

The desired system architecture will follow this diagram:



This architecture uses the AWS cloud platform. This architecture is a desired architecture because to avoid too high costs, the number of instances is reduced in order to remain within the free tier quotas. Typically, we will have one instance for the application and another for the Hadoop part. The process will remain the same, the configurations and links will simply change.

6 Implementation

6.1 Web crawler and scraper for news

6.1.1 Environment

The collection of article data from the chosen sources was done in Python 3 with the help of Scrapy, a framework that allows you to crawl websites and extract the data that interests you.

6.1.2 How it works

To retrieve and collect data from a particular site, "spiders", which are classes that describe how data from that site is explored and retrieved, are developed. This process can be divided into two parts:

- extraction of URL links to explore
- extraction of structured data from an article

For the discovery of links, I browse the pages of one or more categories that group the titles of these articles. For instance, for the articles of *Le Figaro*, the operation can be iterated on the pages given by the link `https://www.lefigaro.fr/politique?page=XXX` by replacing `XXX` by the desired page number. You then just need to write a css selector specific to each site which will automatically extract the URLs.

The method for extracting data from an article is similar: we write a css selector which takes care of retrieving all the paragraphs of an article. You can also add other selectors to retrieve the title, the date or the author. The data is then stored in a JSON file and once all the data have been collected, they are merged into a single csv file that will serve as the dataset to train and test our classification model.

To launch the crawler for a particular site, simply run the following scrapy command:

```
# Implemented spiders: lefigaro, legorafi, lemonde,
# mediaspress, nouvelobs, ripostelaique, francheinfo
mkdir -p <spider_name>_data
scrapy crawl <spider_name>
```

The files are stored inside `<spider_name>_data` folder.

6.1.3 Installation and usage

A README file explaining how to install the crawler is available at the root of the project.

6.2 Model building and Dashboard for Analysis

6.2.1 Environment

The web application UI for analysing a dataset and creating a classification model is developed in React with Recharts for displaying graphs and measures. The interface is served using Apache 2 HTTP server.

The back-end is developed in Python 3 with the help of the web framework FastAPI which allows to design APIs quickly. The back-end takes care of the interaction with HDFS and Hadoop clusters to arrange the data and sequence the MapReduce jobs in the right order. The back-end also interacts with a MySQL database with SQLAlchemy. This database is instantiated and deployed with Amazon Relational Database Service (RDS) to store classification models for future reuse.

Both the front-end and back-end are installed on an Amazon Elastic Cloud Computing (EC2) instance. This instance is generated from an Amazon Machine Image (AMI) that I have previously prepared with softwares required for the deployment and the execution of operations: Hadoop, NodeJS, Python 3, Apache 2, MySQL.

6.2.2 MapReduce jobs

The process to build the classification is divided in 4 steps:

- Preprocessing (vectorisation): Transform an article into a vector through cleaning, stemming and vectorisation
- Class Priors Computation: Compute $P(fake)$ and $P(real)$
- Likelihood Computation: Compute $P(word_i|fake)$ and $P(word_i|real)$ for each word in the vocabulary
- Prediction (testing): Compute articles' posterior probabilities $P(fake|D)$ and $P(real|D)$

The mappers and reducers are written in Python 3 to reduce development cost and are executed with Hadoop Streaming which allows any executable or script to be launched in a MapReduce job.

6.2.3 MapReduce inputs and outputs

The inputs and outputs of the MapReduce jobs are defined as follows:

Preprocessing

- Mapper:
  - input: `(key: id_article, val: text)`
  - output: `(key: id_article, val: word)`
- Reducer:
  - input: `(key: id_article, val: [word1, ...])`
  - output: `(key: id_article, val: [(word1, count1), ...])`

Class Priors Computation

- Mapper:
  - input: `(key: id_article, val: [(word1, count1), ...])`
  - output: `(key: (fake or real, word), val: count)`
- Reducer:
  - input: `(key: (fake or real, word), val: count)`
  - output: `(key: (fake or real), val: prior value)`

Likelihood Computation

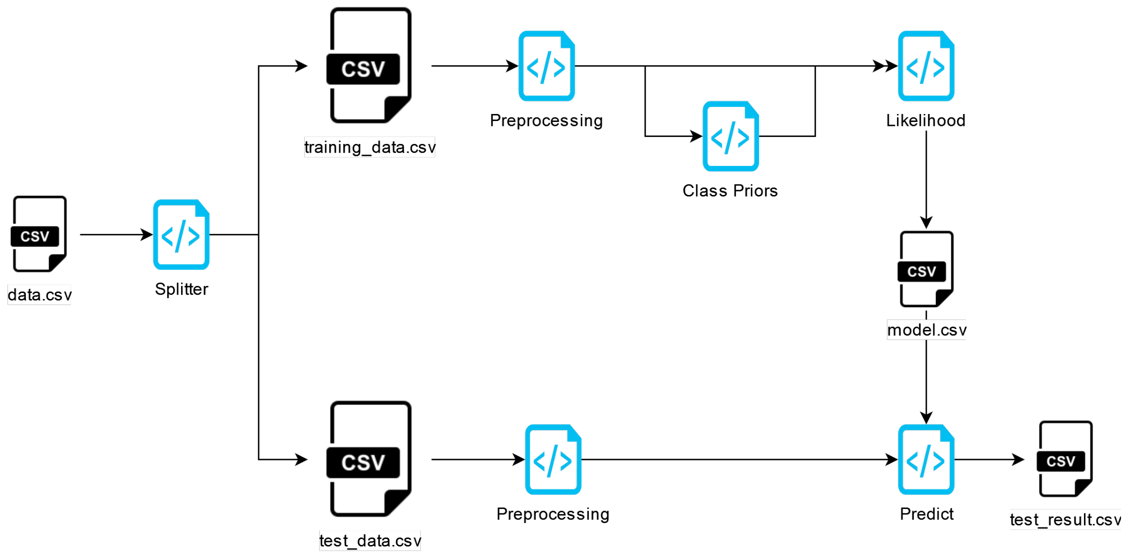
- Mapper:
  - input: `(key: (fake or real), val: [(word1, count1), ...])`
  - output: `(key: word, val: (fake or real, count))`
- Reducer:
  - input: `(key: word, val: [(fake or real, count), ...])`
  - output: `(key: word, val: (real likelihood, fake likelihood))`

Prediction

- Mapper:
  - input: `(key: id_article, val: [word1, ...])`
  - output: `(key: id_article, val: (real likelihood, fake likelihood))`
- Reducer:
  - input: `(key: id_article, val: [(real likelihood, fake likelihood), ...])`
  - output: `(key: id_article, val: (fake or real, real likelihood, fake likelihood))`

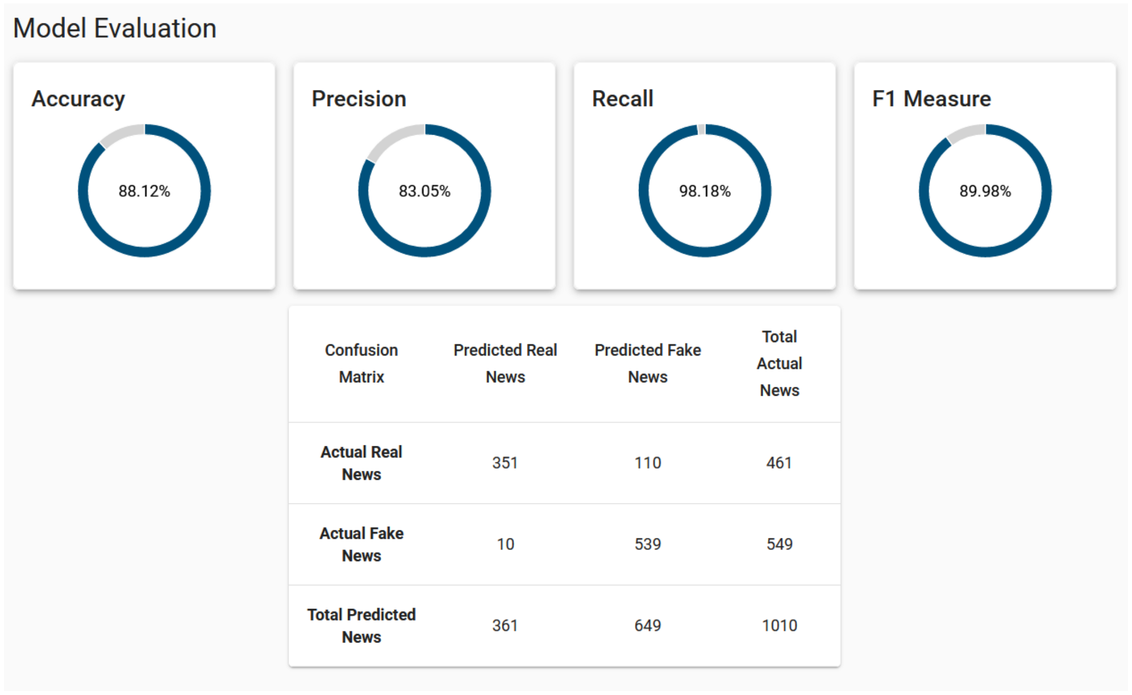
6.2.4 MapReduce jobs sequence

The schema hereunder defines the workflow to build the model through Hadoop MapReduce Jobs:



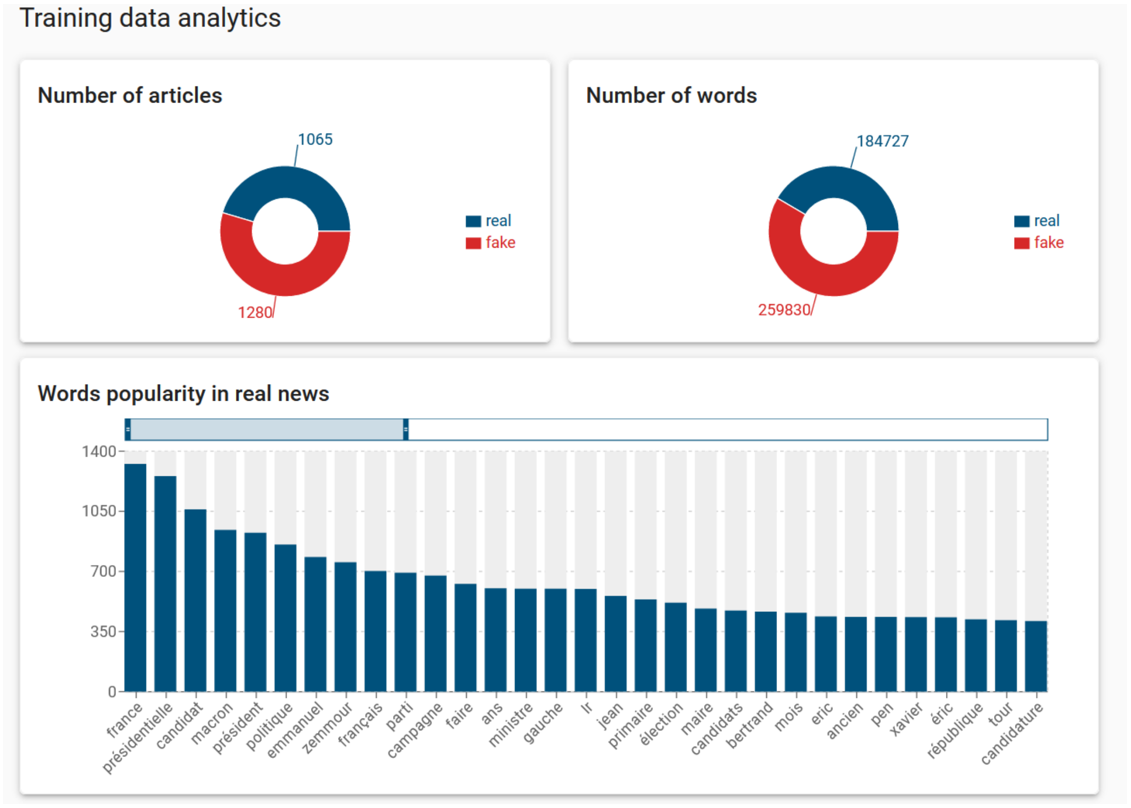
It is worth noting that we can reuse the preprocessing and predict stages to predict any upcoming articles once we have generated our model.csv file.

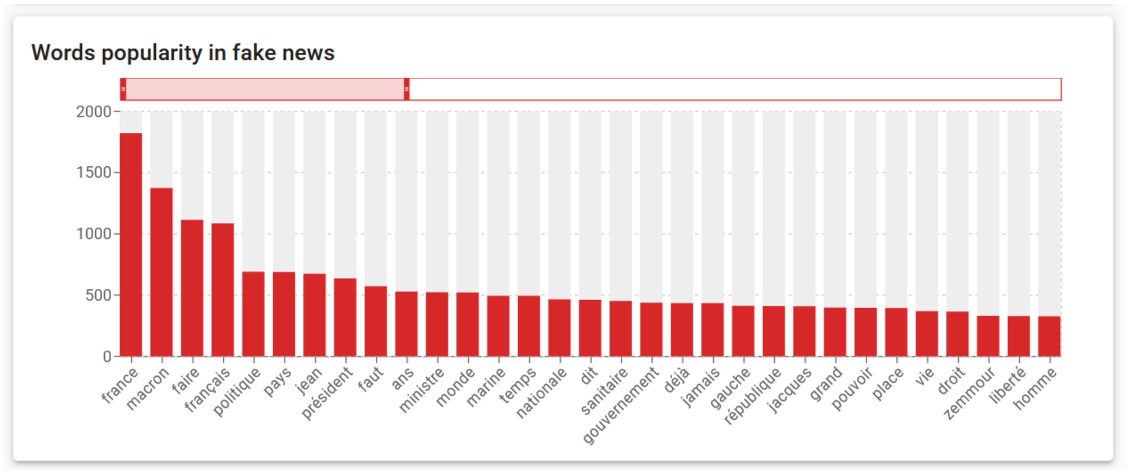
6.2.5 Model Evaluation


Model Evaluation

Accuracy	Precision	Recall	F1 Measure
88.12%	83.05%	98.18%	89.98%

Confusion Matrix	Predicted Real News	Predicted Fake News	Total Actual News
Actual Real News	351	110	461
Actual Fake News	10	539	549
Total Predicted News	361	649	1010





The evaluation gives an accuracy of around 90% and a recall of 98%, which is really good. It should be noted that the number of false positives, i.e. the number of real articles predicted as fake, is quite high compared to the number of true positives. A high rate of false positives is better than a high rate of false positive, since it is better to be vigilant even when faced with a real article. With a high rate of false negatives, it would have been problematic since it prevents further analysis or prevents the reader from being more vigilant.

6.2.6 Installation and Usage

A `README` file explaining how to install the application (front and back) is provided at the root of the projects.

7 Conclusion

In the end, the main requirements of the project were met: design of a supervised learning algorithm and design of a dashboard to analyse and test the prediction model from a dataset.

The main challenge was to transform an existing algorithm into a series of MapReduce jobs while defining the inputs and outputs. Moreover, I had little knowledge of data collection and pre-processing, and I had to learn in the field by informing and documenting myself on the subject. In the end I am rather satisfied with the proposed architecture and growth in natural language processing skills.

For the axes of improvement, one can think non-exhaustively of:

- Deploy the application on the desired architecture described above. For the moment the number of clusters and instances is reduced to avoid too high costs.
- Use a lemmatization algorithm instead of a stemming one for pre-processing stage to get better results (slower and more complex)
- Use another vectorisation method like TfIdf which generally gives better performance
- Implement cross-validation, i.e. several iterations of training and testing on different parts of the dataset to get the best models
- Create a better UI for the dashboard. For the moment, the interface is quite sober and could be more pleasing to the eye
- Create a login and administration system for the generated models. For the moment, the generation of a model generates a token that can be copied/pasted to reuse the model without recalculating it
- Store the data collected by the crawler directly in a NoSQL database (SimpleDB, MongoDB) in the cloud rather than locally
- Categorise more accurately the collected articles by doing factchecking on each article: the generated model seems accurate, but it is possible that real articles are present in an unreliable source and vice versa, which could strongly bias the classification model.