Hate-Speech Recognition based On Twitter Dataset

Simone Dutto, s257348
Data Spaces Course
Politecnico di Torino

March 14, 2020

Contents

1 Introduction

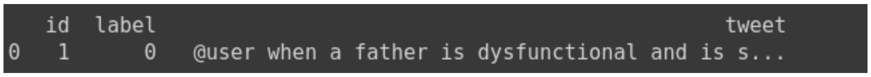
This project is about detecting hate speech in tweets. Nowadays with the spread of social media we cannot rely anymore on manual checking of content, to give an idea according to *liveinternetstats.com* 500 millions of tweets are produced everyday. In addition, we see everyday how hate speech is harming the politic debates and how it can lead to serious consequences in the real world and to the business model of Social Media Platform (Youtube Adpocalypse to give an example).
So, Social Media Platforms are pushing into finding an automatic solution to detect hate-speech content to remove it fast as they can.
Note. This task involves different aspects, from freedom of speech to psychological aspects of our life. We will not dive into them in this project because it is not the topic of the course, but I just want to make clear that I does not think even the most sophisticated classifier will solve the problem, because it is a much more complex problem, but I guess it is a good starting point.

2 Dataset

The dataset is the Kaggle dataset *Twitter Sentiment Analysis.*

2.1 Quantitative Inspection

The training dataset is a csv file containing up to 31962 tweets, with associated label (hate-speech or non hate-speech). The users nicknames are hidden(each mentions was transformed in @user), and hashtags are mantained. The dataset is unbalanced, in fact the 93% of the tweets are considered non hate-speech and there are no missing values.



```
   id  label                                             tweet
0   1      0   @user when a father is dysfunctional and is s...
```

Figure 1: Example of an entry

2.2 Qualitative Inspection

These two images give a perspective on what we are trying to achieve. We would like to define which words and hashtags help the most in finding hate-speech tweets.

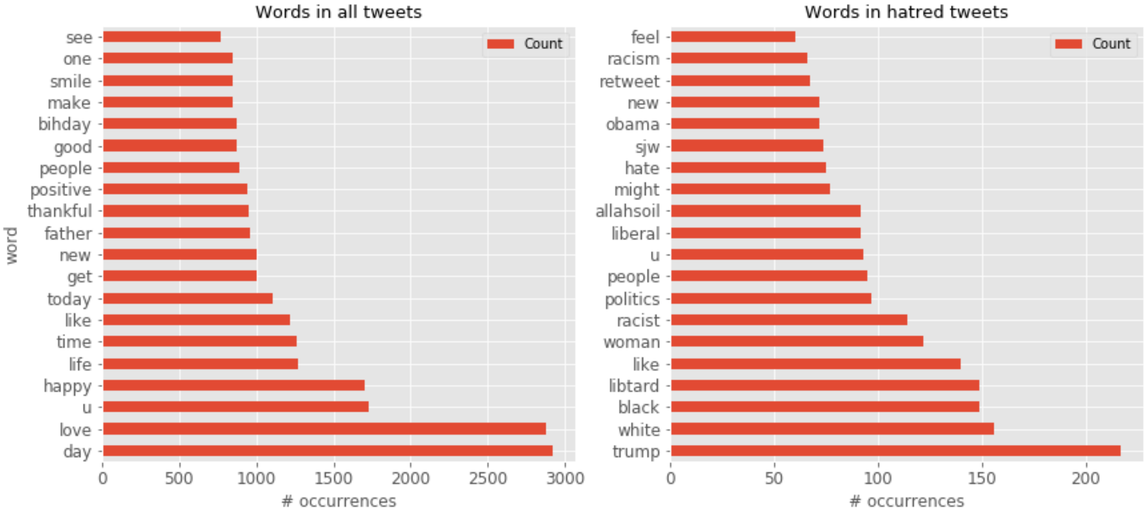


Figure 2: Words occurrences in tweets

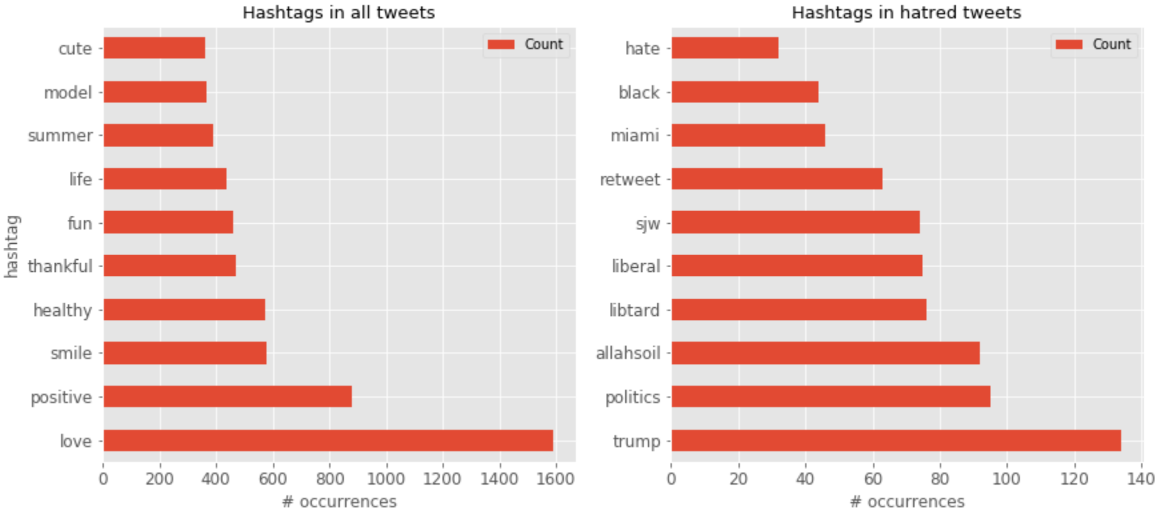


Figure 3: Hashtags occurrences in tweets

3 Preprocessing

To apply classification algorithms to text we need first to transform the text in a vector. The simplest way to think it is to transform each line in a vector long as the dictionary used to compose all the tweets, with one position in the vector to score each word. The simplest scoring method is to mark the presence of words as a Boolean value, 0 for absent, 1 for present. It is clear from this view that it is necessary for computational speed and memory to reduce the vocabulary as much as we can, without loosing content's value.
This process is called normalization.

3.1 Normalization

The pipeline I used to normalize tweet is:

- remove mentions
- convert all to lowercase
- remove character others than alphabetic
- remove stopwords
- lemmatize each word

Removing mentions: Since our dataset is been anonymized mentions do not matter, and to prove it I calculated the correlation between the label and the number of mentions is 0.007. So i decided to remove them.
Converting to lower case: It is a simple operation that reduces the cardinality of the problem without harming the content.
Remove stop-words: Stop-words are words that can be removed without any negative consequences to the final model that you are training. Commonly used in English language include “is”,” and”,” are” etc. The stop-words dictionary can be expanded to add specific words, for example I removed also the word ”amp”, because its presence was due to the data extraction method and even if it was widely present it does not mean anything.
*Lemmatize each word:*Lemmatization aims to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma (Ex. Cats -¿ Cat).
To give an idea of how important is this operation, here is the same Words Occurrences Graph in Figure 2 without using normalized tweets.

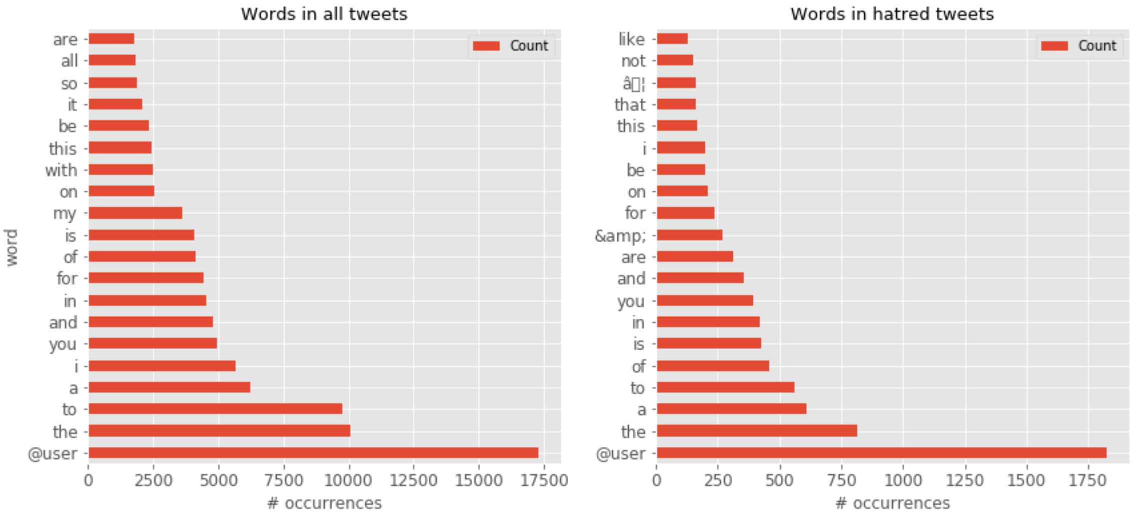


Figure 4: Words occurrences in tweets without normalization

In addition the number of unique words goes from 67223 to 35127. This is a huge improvement, because not only we are extracting the most useful words from a text but we are also reducing the cardinality of the problem, so our problem becomes easier to tackle.

3.2 Scoring each words

The first approach I explained to score each word(1:present 0:absent) is called Bag of Words, it is really simple, but not very clever. Because it does not consider the frequency of term, for example if there are multiple occurrences of a word in a document its score should be high, or if a word is in each document of the corpus(the entire dataset) probably is not carrying much importance, so its score should be low.
For this reason, it is commonly used the TF-IDF.
TF-IDF: Term Frequency * Inverse Document Frequency is a weighting function where the score is calculated upon:

- Term Frequency: how often a word appears in a document, divided by how many words there are

- Inverse Document Frequency: the total number of documents divided by number of documents with term t in it. It expresses how rare is a word in the corpus

A High score in TF-IDF is reached by a high term frequency(in the given document) and a low document frequency of the term in the whole collection of documents.
Note. The tfidfVectorized in ntlk library is represented with a Sparse Matrix, so it is memory efficient.

3.3 Upsampling

The dataset is composed by 29720 non hate-speech tweets and 2242 hate-speech tweets. At this state if we train a model on imbalanced data, the results will be misleading. In such datasets, due to lack of learning, the model would simply predict each tweet as a good tweet. So we need to balance the dataset, I used *resample()* of sklearn to have 29720 and 29720 of both classes.

3.4 Grammar Evaluation

I tried to use a library called *gingeit* to evaluate the number of grammar errors in each tweet, because I thought the number of errors could have been correlated to hate-speech because a person usually makes more mistakes if he is angry.
It turned out that, also due to the library which is not really fine-tuned for slang, 996 out of 1000 tweets have at least one grammar error. Therefore there is no correlation between errors and label, so I decided not to include this column in the input.

4 Models

In this section I will analyze the model I tried. Since accuracy is really high (95%) I will try to evaluate models also on different aspects:

- Accuracy/Recall
- Training time
- Evaluation time
- Possibility of online learning

Accuracy/Recall It is important to evaluate correctly the metric we want to highlight. Recall is the fraction of the total amount of relevant instances that were actually retrieved. So it is very important, because if the tweet has to be manually inspected after the first automatic detection before being deleted, the manual system must not be overloaded with a lot of non hate-speech tweets.

Training time My dataset is a toy example, but in real-world application the training dataset is huge (remember 500 millions tweets per day), so it is important to have the fastest training model possible.

Evaluation time The time to detect a hate-speech must be really fast, so hate-speech can be deleted fast and it cannot harm the platform.

Possibility of online learning Once a model is established people will try a way to fool it really fast (Example. sex → $3x), so it is important to be able to update constantly the model, without having to wait the entire training time each time.

4.1 Naive-Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

If we exploit the naive condition, independence between variables, we have:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

We have $P(x_1, \dots x_n \mid y)$ and we evaluate the probability distribution with a MAP approach.

$$\begin{gathered} P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \\ \Downarrow \\ \hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y), \end{gathered} \tag{1}$$

This is a generative, so the posterior distribution is estimated calculating $P(X|Y)$ and $P(Y)$. It is an offline algorithm, so it is not possible to update the model simply by adding an entry, if we add an entry we need to recalculate all the probabilities.

4.2 Logistic Regression

Logistic Regression is a learning algorithm based on the so-called Logistic Function to evaluate the probability of belonging to a class:

$$f(x) = \frac{1}{1 + e^{w_0 + \sum_i w_i * X_i}}$$

We use Gradient Ascent to update the weights:

$$J(\mathbf{w}) = \sum_{i=1}^{m} -y^{(i)} log\Bigg(\phi\big(z^{(i)}\big)\Bigg) - \big(1 - y^{(i)}\big) log\Bigg(1 - \phi\big(z^{(i)}\big)\Bigg)$$

$$\Delta w_j = -\eta \Bigg(\frac{\partial J}{\partial w_j}\Bigg)$$

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}$$

Logistic Regression is a discriminitive approach, and it is possible to update the model online each time we want to add an entry. In fact, it is an optimization problem. It is fast to train and fast to evaluate, and it reaches good performances in the limit.

4.3 Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. To create the rules the algorithms usually work top-down, by choosing a variable at each step that best splits the set of items. Different algorithms use different metrics for measuring ”best”. I used Gini Impurity to evaluate how good is a split:

$$Gini : Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

Random Forest is usually really accurate, and it can shape non-linear boundaries really well, in fact we peak at 1.00 with this model. However it is really slow both to train and also to evaluate an entry.

4.4 Qualitative Model Evaluation

I will go through a hyperparameters selection in the next section, but the table below gives a rough idea on how models perform on the data.

Models Comparison			
Model	Training Time	Test Time	Accuracy
Naive-Bayes	0.019s	0.009s	95%
Logistic Regression	0.583s	0.004s	97%
Random Forest	17.3s	2.27s	99%

Table 1: Models comparison

5 Experiment

In the section before I went through the models I used for my experiment, but before analyzing the results I will explain my validation process.

5.1 Leave-One-Out Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models, such as it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. The dataset is splitted in k folds (5 in my experiments), the model is trained upon k-1 folds and tested on the one-out. This is done for k times, and the results are averaged.

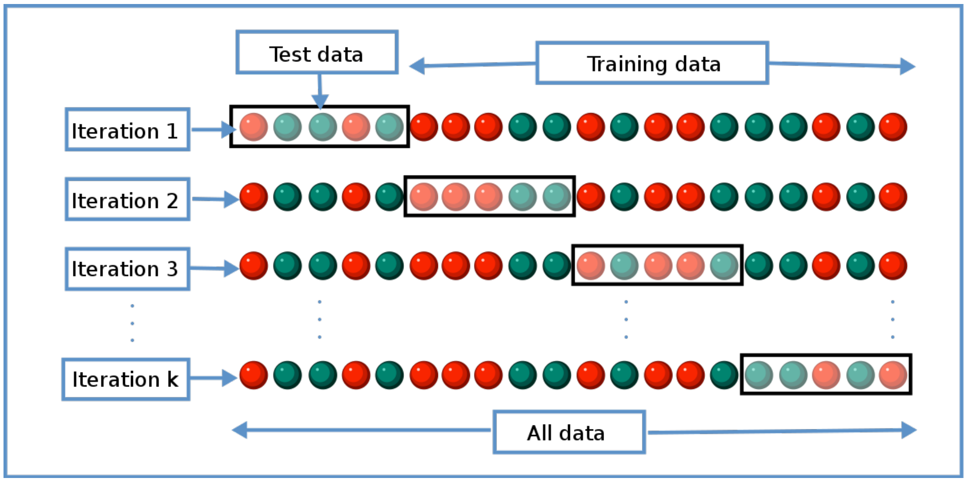


Figure 5: k-Fold Cross Validation

5.2 GridSearch

For a fine-tuned selection of the models I used a GridSearch Approach. It generates all the possible combination from a list of hyperparameters. In this way I can choose properly the hyperparameters and test the model on the test set to evaluate performance of each model. **Note.** The function *GridSearchCV()* of *sklearn* implements both the strategy, GridSearch in combination with k-fold cross-validation. In the next section I will go though the hyperparameters I decided to tune and how they affect the models.

5.3 Naive-Bayes

For the Naive-Bayes I decide to tune *alpha*. Alpha is used to do smoothing or rather Laplace smoothing. The basic idea is to increase the probabilities of all bigrams in your non-maximum likelihood equation by alpha to make everything non-zero. So at testing time, if the model encounters word which is not present in train set then its probability of existence in a class is zero, the probability will be $0+\epsilon$. The higher alpha the noisier is the output, but it is actually useful when train set and test set are really different.

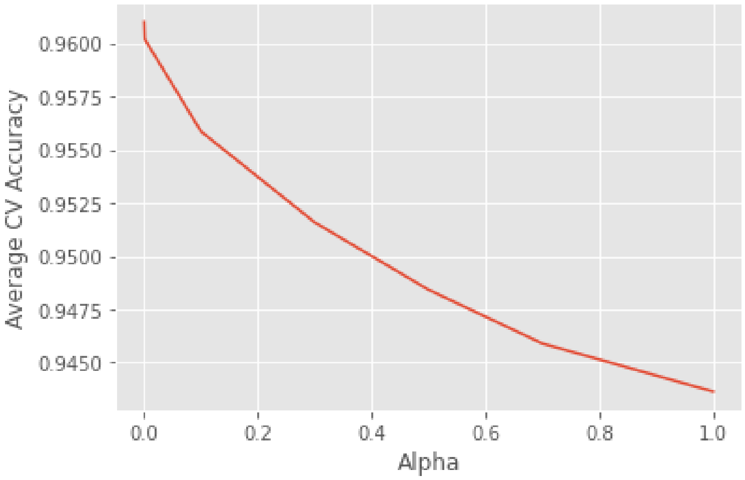


Figure 6: Naive-Bayes Cross Validation

What I can guess from my experiment is that train set and test set have the same dictionary, so alpha greater than 0 creates just noise and no actual value.

5.4 Logistic Regression

For Logistic Regression I choose to optimize C. It is a regulation parameter, which is a term used to enforce or diminish the weight update.

$$L2 : \frac{\lambda}{2}\|\mathbf{w}\|_2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

C is the inverse of λ and it is basically a “balance” to the learning process - between regulating too much (forcing faulty convergence to local parameter space and no capacity to move to better solutions OR converging to an extremely precise area - that leaves with overfitting) and regulating too little (Where you effectively fail to regulate signal parsing - and cannot ascertain convergence parameters - such as normality conditions). The higher C the lower is the regularization.

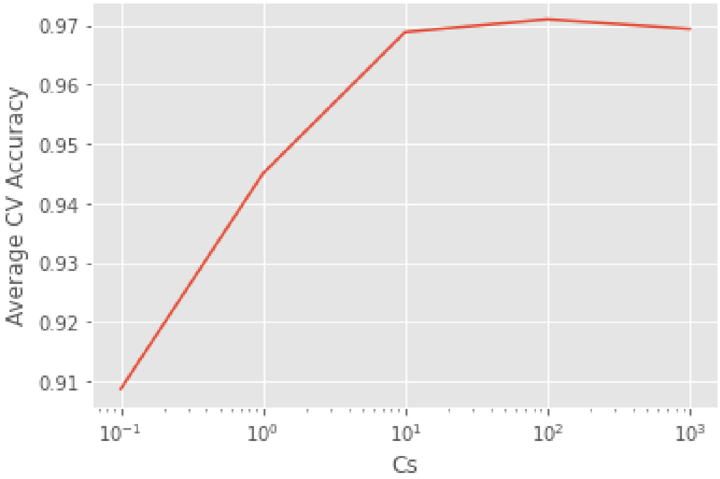


Figure 7: Logistic Regression Cross Validation

It is easy to see that a little bit of regularization helps the optimization to overcome local minima.

5.5 Random Forest

To optimize the Random Forest I choose 2 hyperparameters:

- Number of estimators: The number of tree in the forest, the higher the less noisier is the model and the classifier performs really well
- Maximum number of features: It is the number of features to consider each time to make the split decision, it helps preventing overfitting.

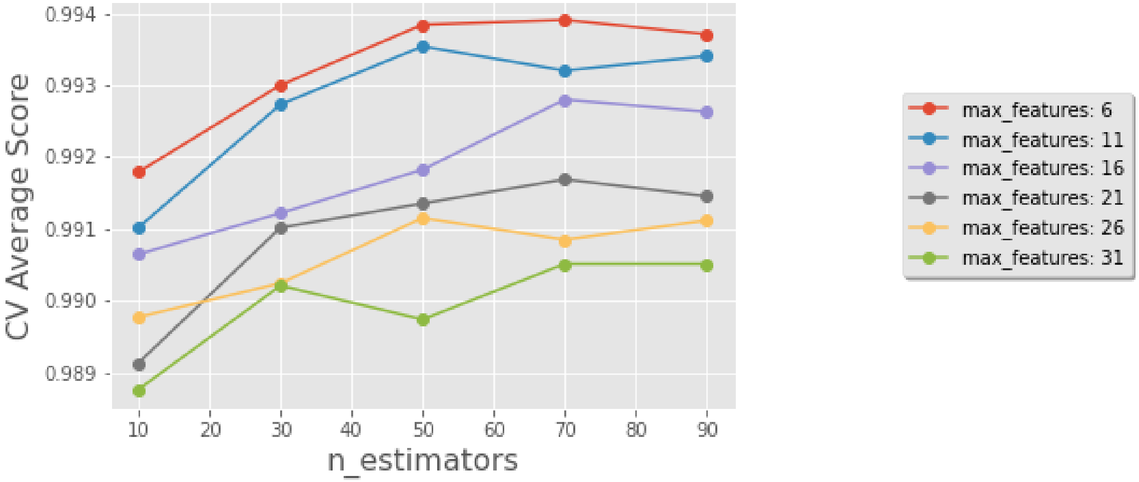


Figure 8: Random Forest Cross Validation

I tried to limit the maximum depth of the decision trees to 200 to fasten the train and test time. But it affects too much the performance (82% accuracy) because we have around 35k features, so the decision trees have a lot of splits to do and so they are very deep.

5.6 Performances on Test Set

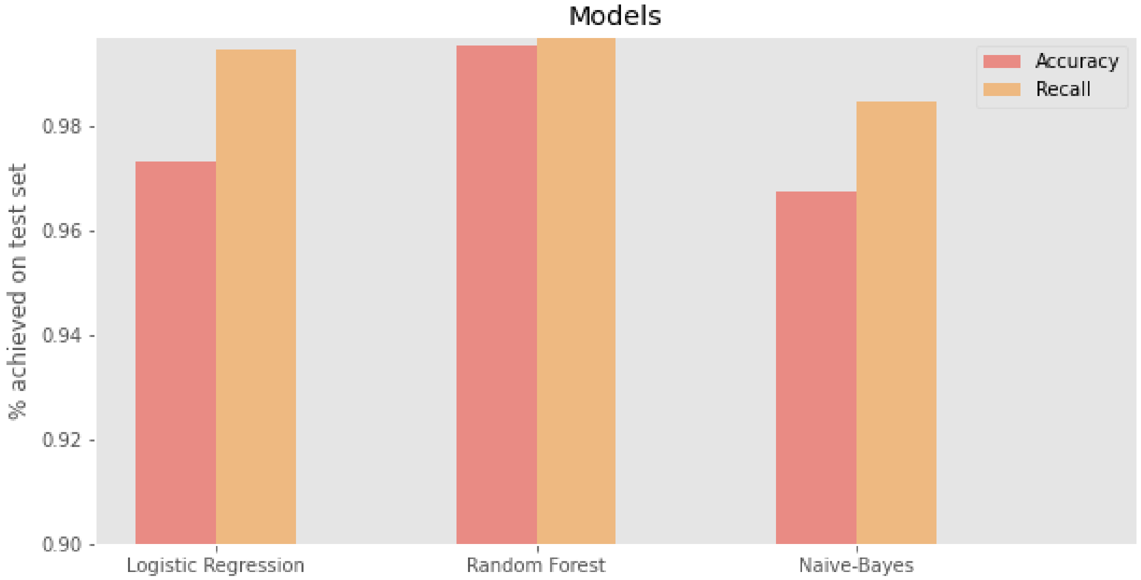


Figure 9: Performance on Test Set

6 Conclusion

In conclusion, if I were a company I would use the Logistic Regression model. Because: it is not the quickest but way faster than random forest to train, it is the faster to evaluate the test set, it achieves optimal recall 99% and it can be updated each time the company change the Terms of Service. A consideration I would like to highlight is that these models achieve very high accuracy even if the task can be considered really complex also for human, but there is a big problem that our model cannot solve. In fact, the person (or group of people) which decides whether or not a tweet is hate-speech could bias the model. So we build very specific hate-speech detection, and sometimes it can be tricky to decide where is the line, and this line can affect a lot the ecosystem of a platform if some kinds of tweets are systematically banned.
Final Note on the Code. I developed my models and made my experiments using Google Colaboratory and Python with sklearn library. The Python Notebook can be find at *this link*.