

*Stance detection on tweets about the covid-19 vaccination topic*

Project of the Data Mining course 2020/2021

*Edited by:*

*Castrignano Matteo*

*Dallatomasina Erica*

Sommario

[1. Introduction 2](#_Toc74243433)

[2. Analysis 2](#_Toc74243434)

[2.1. Use case diagram analysis 2](#_Toc74243435)

[3. Workflow 3](#_Toc74243436)

[3.1. Data collection 3](#_Toc74243437)

[3.2. Data pre-processing 4](#_Toc74243438)

[3.3. Text representation: tokenization, stop-word filtering and stemming 5](#_Toc74243439)

[3.4. Supervised learning stage 6](#_Toc74243440)

[3.4.1. Results for n-gram = (1,1) 7](#_Toc74243441)

[3.4.2. Results for n-gram = (1,2) 8](#_Toc74243442)

[3.5. Classification of unlabelled tweets 10](#_Toc74243443)

[3.6. Concept drift analysis 10](#_Toc74243444)

[4. Implementation 11](#_Toc74243445)

# Introduction

Our goal was to create an application that can analyze tweets to discover Italian people's opinion about anticovid-19 vaccines. In particular, the tweets that regard the vaccination topic are scraped according to some searching criteria (like some keywords and the date) and then they are automatically labeled in one of three classes (negative, positive, or neutral) depending on the stance expressed in the tweet itself. Finally, the application shows a pie-chart that plots the number of tweets for each class.

# Analysis

The application is thought for an expert user. He can set the period about which he is interested in and also the vaccine he is interested in. If he does not specify anything, the application will analyze all the tweets from the 1st of February 2021 without searching for a specific vaccine.

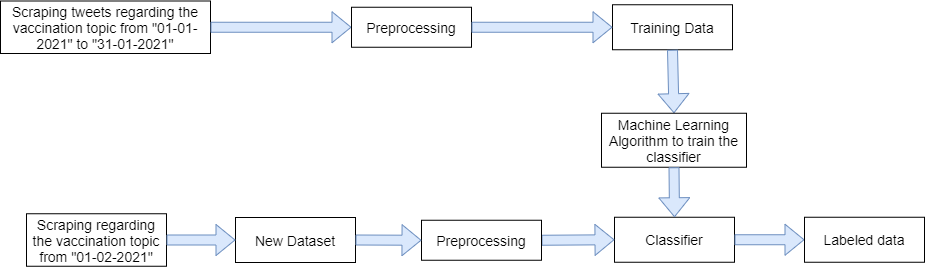
# Use case diagram analysis

# Workflow

In order to perform the stance detection on the tweets, it was necessary to train an appropriate classifier (**supervised learning stage**). Then, using the classifier learned in this previous phase, we can apply it to the new unlabeled tweets in order to classy them (**unsupervised learning stage**).

To apply classification algorithms on the tweets and to construct the classifier, the tweets must pass through a **preprocessing phase** in which they are cleaned from all the features that are not useful for the classification task. Then each tweet must also be transformed into a numerical vector (**feature representation**).

The following image shows the workflow adopted in order to obtain the classifier and to use it for the classification of new tweets:



In the following we will explain more in detail the meaning of each phase.

# Data collection

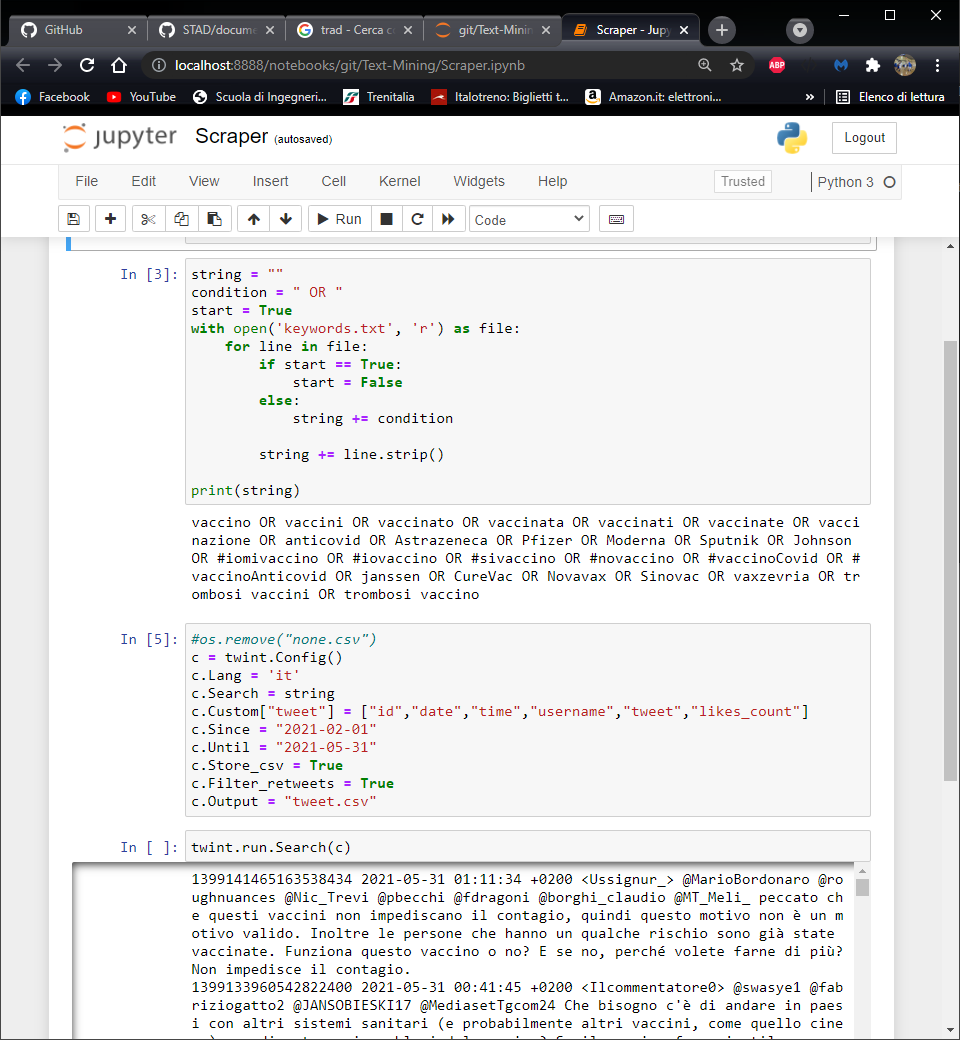
The raw tweets are scraped from Twitter using the *Twint[[1]](#footnote-1)* library. Using the functions offered from this library, we can set some options in order to scrape tweets according to some parameters. In particular we set:

* The date, in order to scrape tweets posted only on a specific period.
* The language, in order to scrape only Italian tweet.
* Some keywords to scrape the tweets only related to the vaccination topic.

The scraped tweets are characterized by the following attributes (whose name is self-explanatory):

* id
* date
* time
* username
* tweet
* likes\_count

In the following image is shown the code used for scraping. In the file “keywords.txt” are contained the keywords that are used linked with the *OR* conjunction in order to scrape tweets related to the vaccination topic. We tested different sets of keywords and we choose the one that gives the better results in terms of scraping (so the one that allows to scrape more tweets related to the vaccination topic).



# Data pre-processing

In the pre-processing phase raw tweets are elaborated in order to clear them. In particular, we do the following steps:

* Remove all the links, emoji and images contained in the tweets
* Remove all the hashtag symbols (#) from the tweets. For example, if a tweet contains “#word” we transform it in “word”. We decided to maintain the word associated to the hashtag because it is often important to understand the stance of the tweet (for example there can be hashtags like #iomivaccino or #vaccinomorte that are indicative of the opinion about vaccines expressed in the tweet)
* Remove all the mentions to other users in the tweet
* Put the text of the tweets in lower case
* Remove all the punctuation
* Remove all the extra-spaces

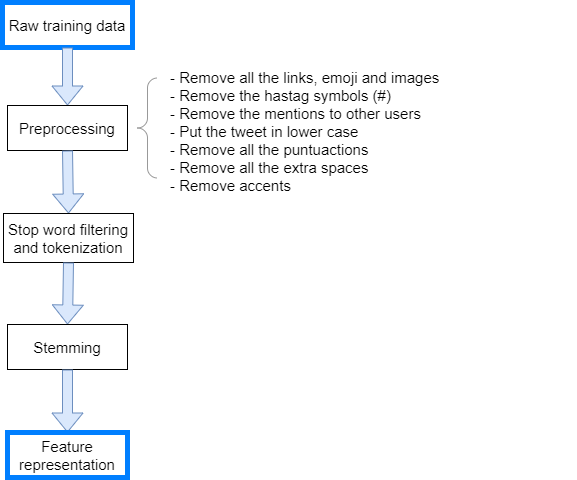
As an example, the tweet*"@valy\_s È una cosa normale. L'immunità di gregge è a senso unico. Se vaccini tutti non sconfiggerai mai la malattia. Quando lo capiranno, torneremo a vivere normalmente! Se vuoi fermare la pandemia, studiala e crea una medicina. Il vaccino serve a poco!"* is transformed into *“e una cosa normale l immunita di gregge e a senso unico se vaccini tutti non sconfiggerai mai la malattia quando lo capiranno torneremo a vivere normalmente se vuoi fermare la pandemia studiala e crea una medicina il vaccino serve a poco”.*

# Text representation: tokenization, stop-word filtering and stemming

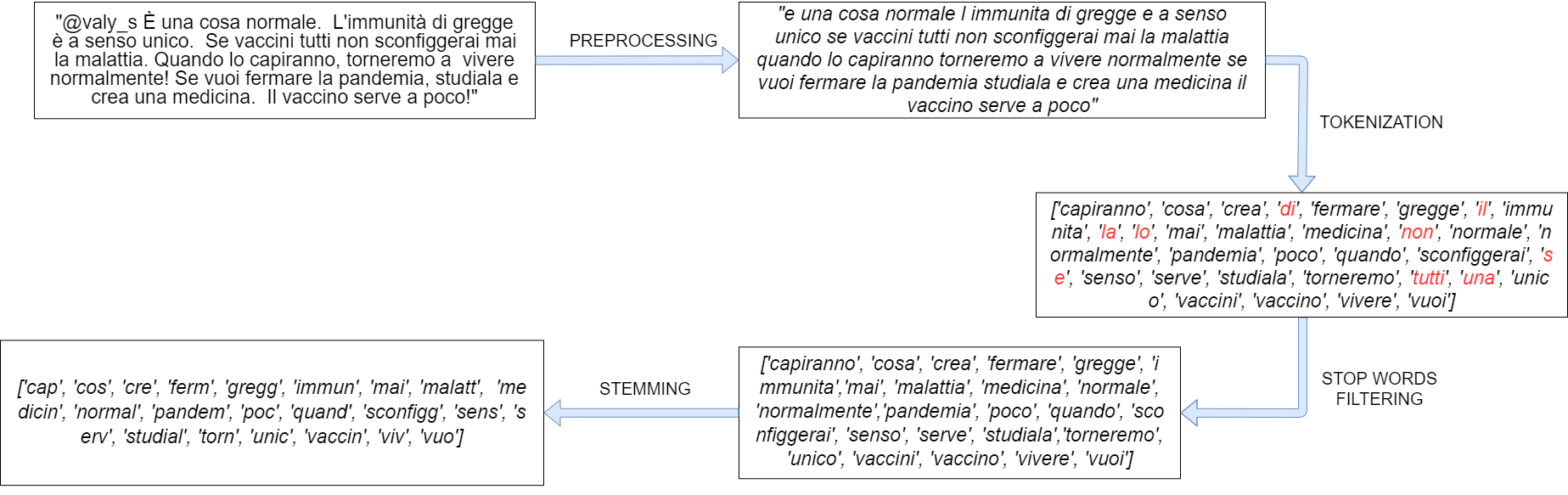
In order to apply classification algorithms to the tweets we have to represent them as numerical vectors. In particular we have to follow the following steps:

* Extract tokens from tweets (**tokenization phase**). We use the BOW representation testing different values of n-gram (we consider only unigrams and unigrams and bigrams together)
* Remove stop-words (**stop-word filtering**) from the set of extracted tokens
* Eventually perform a **stemming phase**, in order to reduce each token to its “root form”
* Perform a **tf-idf transformation**, where the importance of each stem is computed using the *Inverse Document Frequency*

The following image shows all the phases we carried out to transform each tweet into a numerical vector. For clarity, also the pre-processing phase is considered.



In the following image is shown an example of how a tweet is transformed when it passes through each one of these phases:



# Supervised learning stage

In the supervised learning stage, the classifier is learned from a training set. In particular, the training set is constructed by scraping tweets from “01-01-2021” to “31-01-2021” and manually labelling 906 instances (302 instances for each class).

Then we trained different models and we tested them using the cross-validation procedure[[2]](#footnote-2). Finally, the classifier that gave us better results (in terms of accuracy, precision and recall) was chosen as the final classifier to be adopted for the classification of tweets.

To train the model different experiments were performed, considering different classifiers and different text representations techniques. We decided to test 7 different classifiers:

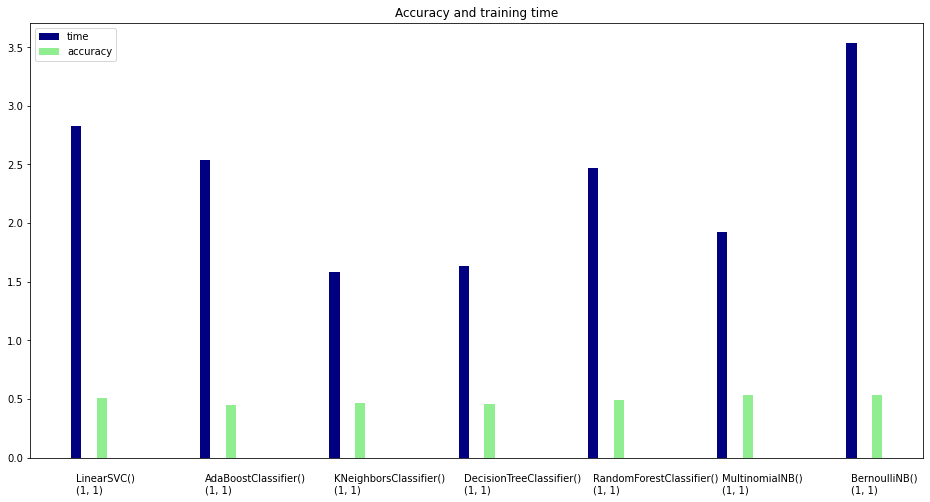
* Multinomial classifier
* Bernoulli classifier
* Linear support vector classifier
* Random forest classifier
* K-nn classifier
* Adaboost classifier
* Decision tree classifier

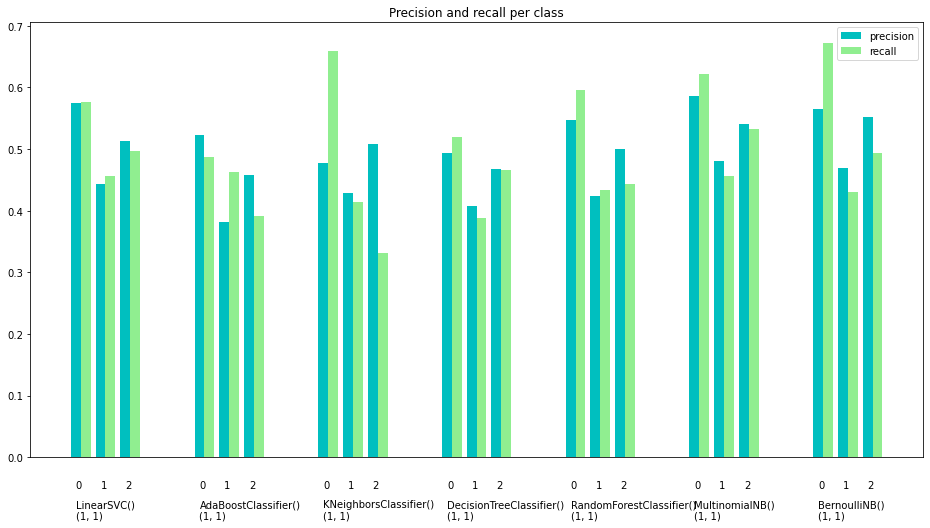
We made some experiments in which we defined different sets of stop-words, but we saw that the better results were obtained by using the set of Italian stop-words offered by *Snowball Taurus[[3]](#footnote-3)*. We also try to do some experiments without the stemming phase, but better results are achieved by adopting the stemming phase. For the sake of brevity, we will report only the results obtained with the experiments performed with the *Snowball Taurus* Italian stop-words list and performing the stemming phase.

In the following, we report the results obtained testing the different classifiers using the tokenization with only unigrams (n-gram = (1,1)) and with unigrams and bigrams (n-gram = (1,2)). In each one of these experiments, we considered as stop-words the *Snowball Taurus* Italian stop-words list, and we performed the stemming phase. We also report some histograms that show the overall accuracy and the trainining time for each classifier and for each classifier, the precision, the recall and the f1-score for each one of the three classes.

## Results for n-gram = (1,1)

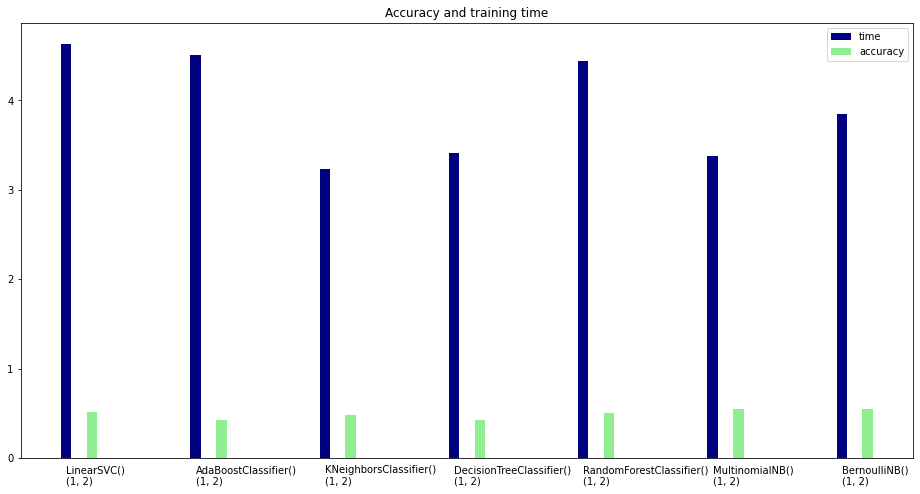
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Classifier* | *Class* | *Precision* | | *Recall* | *F-score* | | *Accuracy* | *Time (s)* | *Confusion matrix* |
| BernoulliNB() | *0* | *0.57* | *0.67* | | | *0.61* | *0.53* | *1.92* | *203 60 39*  *90 130 82*  *66 87 149* |
| *1* | *0.47* | *0.43* | | | *0.45* |
| *2* | *0.55* | *0.49* | | | *0.52* |
| MultinomialNB() | *0* | *0.59* | *0.62* | | *0.6* | | *0.53* | *1.78* | *188 67 47*  *74 138 90*  *59 82 161* |
| *1* | *0.48* | *0.46* | | *0.47* | |
| *2* | *0.54* | *0.53* | | *0.54* | |
| LinearSVC | *0* | *0.57* | | *0.58* | *0.58* | | *0.51* | *2.42* | *174 77 51*  *73 138 91*  *56 96 150* |
| *1* | *0.44* | | *0.46* | *0.45* | |
| *2* | *0.51* | | *0.49* | *0.51* | |
| RandomForestClassifier() | *0* | *0.56* | *0.58* | | *0.56* | | *0.51* | *3.09* | *176 79 47*  *85 135 82*  *56 98 148* |
| *1* | *0.43* | *0.45* | | *0.44* | |
| *2* | *0.53* | *0.49* | | *0.51* | |
| KNeighborsClassifier()  N = 5 | *0* | *0.48* | *0.66* | | *0.55* | | *0.47* | *1.76* | *199 70 33*  *113 125 64*  *105 97 100* |
| *1* | *0.43* | *0.41* | | *0.42* | |
| *2* | *0.51* | *0.33* | | *0.4* | |
| AdaBoostClassifier | *0* | *0.52* | | *0.49* | *0.5* | | *0.45* | *2.56* | *147 99 56*  *78 140 84*  *56 128 118* |
| *1* | *0.38* | | *0.46* | *0.41* | |
| *2* | *0.46* | | *0.39* | *0.42* | |
| DecisionTreeClassifier() | *0* | *0.5* | | *0.52* | *0.51* | | *0.45* | *2.53* | *158 74 70*  *91 118 93*  *68 102 132* |
| *1* | *0.4* | | *0.39* | *0.4* | |
| *2* | *0.45* | | *0.44* | *0.44* | |

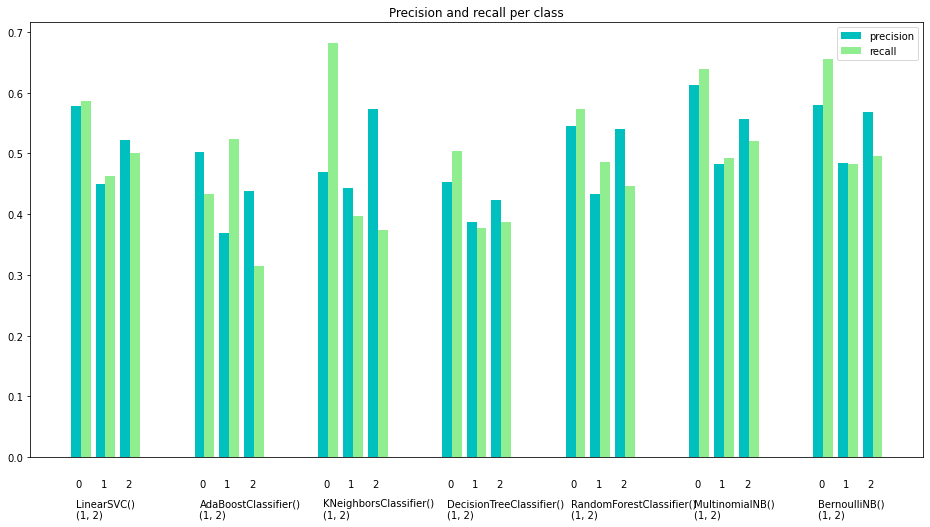




## Results for n-gram = (1,2)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Classifier* | *Class* | *Precision* | | *Recall* | *F-score* | | *Accuracy* | *Time (s)* | *Confusion matrix* |
| BernoulliNB() | *0* | *0.58* | *0.66* | | | *0.52* | *0.55* | *3.84* | *198 66 38*  *80 146 76*  *63 89 150* |
| *1* | *0.49* | *0.48* | | | *0.48* |
| *2* | *0.57* | *0.5* | | | *0.53* |
| MultinomialNB() | *0* | *0.61* | | *0.64* | *0.63* | | *0.55* | *3.38* | *193 72 37*  *65 149 88*  *57 88 157* |
| *1* | *0.48* | | *0.49* | *0.49* | |
| *2* | *0.56* | | *0.52* | *0.54* | |
| LinearSVC | *0* | *0.58* | | *0.59* | *0.58* | | *0.52* | *4.63* | *177 77 48*  *72 140 90*  *57 94 151* |
| *1* | *0.45* | | *0.46* | *0.46* | |
| *2* | *0.52* | | *0.5* | *0.51* | |
| RandomForestClassifier() | *0* | *0.55* | *0.57* | | *0.56* | | *0.5* | *4.44* | *173 89 40*  *80 147 75*  *64 103 135* |
| *1* | *0.43* | *0.49* | | *0.46* | |
| *2* | *0.54* | *0.44* | | *0.49* | |
| KNeighborsClassifier()  N = 5 | *0* | *0.47* | | *0.68* | *0.56* | | *0.48* | *3.23* | *206 67 29*  *127 120 55*  *105 84 113* |
| *1* | *0.44* | | *0.4* | *0.42* | |
| *2* | *0.57* | | *0.37* | *0.45* | |
| DecisionTreeClassifier() | *0* | *0.45* | *0.5* | | *0.48* | | *0.42* | *3.41* | *152 85 65*  *94 114 94*  *89 96 117* |
| *1* | *0.39* | *0.38* | | *0.38* | |
| *2* | *0.42* | *0.39* | | *0.4* | |
| AdaBoostClassifier | *0* | *0.5* | | *0.43* | *0.47* | | *0.42* | *4.51* | *131 114 57*  *79 158 65*  *51 166 95* |
| *1* | *0.37* | | *0.52* | *0.43* | |
| *2* | *0.44* | | *0.31* | *0.36* | |





As we can see from the previous tables in both cases of n-gram = (1,1) and n-gram = (1,2), the best overall accuracy is reached by the Multinomial classifier and the Bernoulli classifier. The better overall accuracy is equal to 55%. This is probably because in the tweets sarcasm and irony are often used, and this makes the classification task more complex. Moreover, there can be cases in which also for a human is difficult to assign a class to a tweet, due to the fact that the expressed opinion is not so much clear.

We decided to consider only unigrams. In fact, considering unigrams and bigrams do not improve so much the classifiers (with respect to the case in which we consider only unigrams) but instead the training time is almost doubled.

For the final classifier we decided to adopt the Bernoulli classifier (aggiungiamo una spiegazione più dettagliata?). più dettagli cosa??

# Classification of unlabelled tweets

In this phase tweets from “01-02-2021” are classified using the Bernoulli classifier trained in the supervised learning stage, after passing through the pre-processing phase and the text representation phase.

Qui vogliamo riportare un po’ di risultati? Uno per classe

# Concept drift analysis

Since the lexicon used by Twitter’s users can change over time, a concept drift analysis must be carried out. If the presence of concept drift is detected, the model must be updated in order to reduce the effects of the concept drift.

For analysing the presence of concept drift, we adopted the following strategy (in the following with “initial training set” we will refer to the training set made up of 906 tweets from January and with “initial classifier” to the classifier trained using the initial training set):

* we manually labelled 60 tweets (20 tweets for each class) of the months of February, March and April. For each month we sort the tweets in descending order using the number of likes they received, and we classify them starting from the one with the higher number of likes and going on until there were 20 tweets per class.
* Then we test the initial classifier on each one of the labelled sets of February, March and April.
* Then a new classifier was constructed using the initial training set and the 60 labelled tweets from February. This new classifier was tested on the tweets of March and April.
* We compared the scores of the initial classifier and the new classifier. If the scores of the new classifier are better than the initial one, it means that there is concept drift.
* The same procedure can be repeated constructing a new classifier considering the initial training set and the labelled tweets from March and testing it using the tweets from April.
* Dobbiamo aggiungere anche i risultati del concept??

# Implementation

The whole application is implemented in *Python* using the *Jupyter* notebook. For implement the machine learning algorithms we exploit the *SK-learn* library.

The application consists of different modules. The most important are:

* Scraper.ipynb 🡪 in which there is the code for scraping tweets
* Preprocessing.ipynb 🡪 in which there is the code for the pre-processing
* CrossValidationFinal.ipynb 🡪 in which there is the code for perform the cross-validation
* TrainingModel.ipynb 🡪 in which the chosen model is trained and saved in the file “InitialModel.pkl”
* Utilities.ipynb 🡪 that contains some utilities functions
* ConceptDrift.ipynb 🡪 in which the concept drift analysis is implemented
* UtilitiesConceptDrift.ipynb 🡪 that contains some utilities functions for the concept drift
* Application.ipynb 🡪 is the module from which the user can invoke the classification of tweets, setting the period he wants to analyse and the name of the vaccine he is interested in

The code is available on this GitHub repository: https://github.com/Matteo-Castrignano/Text-Mining

1. All information about *Twint* can be retrieved from GitHub repository: <https://github.com/twintproject/twint> [↑](#footnote-ref-1)
2. The experiments were performed using a 5-fold cross validation procedure. Using a 5-fold cross validation procedure, at each iteration the model is constructed and tested using 60 instances as test set, and the remaining instances as training set. [↑](#footnote-ref-2)
3. The list of Italian stop words can be retrieved here: <http://snowball.tartarus.org/algorithms/italian/stop.txt> [↑](#footnote-ref-3)