
# UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Data Mining and Machine Learning

## *News Classifier*

Project Documentation

---

*AUTHOR:*
Leonardo Bargiotti

Academic Year: 2022/2023

# Contents

# List of Figures

# 1 — Introduction

Unstructured data in the form of text: chats, emails, social media, survey responses is present everywhere today. Text can be a rich source of information, but it can be hard to extract insights from it. Text classification is one of the important task in supervised machine learning (ML). It is a process of assigning tags/categories to documents helping us to analyze automatically text in a cost-effective manner. It is one of the fundamental tasks in Natural Language Processing with broad applications such as sentiment-analysis, spam-detection, topic-labeling, intent-detection etc.

This project exploits text mining in order to automatically categorize news articles to their right topic (for example sport, business, world and sci/tech). This application could be used to classify text of other subject (not only news), depending on which dataset is given in input. [1]

## 1.1 Goals

The aim of this paper is to explain the choices and the strategies adopted on the project and development of **News Classifier**. In order to accomplish it, the first step is perform preprocess for having a suitable dataset and then it is used several classifiers, to determinate which predicts the right class.

## 1.2 Initial Dataset

In order to realize this application are used two different dataset:

1. **AG News** [2]
2. **BBC News** [3]

### 1.2.1 AG News

The first dataset is composed by 120000 rows in training set and 7600 in testset. It is perfectly balanced in fact it has four class *World, Business, Sports and Sci/Tech* and each one have the same number of instances, in particular 30000 in training set and 1900 in test set. It has three columns: one relative to the class of the news, the second is the title of the news and the last one is the news. For this application the title is not useful. The column, relative to classes, contains numbers associate to the four classes *1,2,3,4 are associate respectively with World, Sports, Business and Sci/Tech class.*

### 1.2.2 BBC News

In the second one is smaller than previous, it has only 2225 rows in dataset (it will be divide into training and test set using *train_test_split* [4], default value of *test_size* is 0.3). It has five classes *Sport, Business, Tech, Politics and Entertainment* and they are not balanced, the class most frequent is Sport with 511 instances and the class minus frequent is Entertainment with 386. The are only two columns: one relative to the description of the news and the other to the corresponding class. The column, relative to classes, contains strings that they will be encode

[1] GitHub repository for the project: `GITHUBURL`

[2] Link for AG News Dataset from Kaggle: `https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset`

[3] Link for BBC News Dataset from Kaggle: `https://www.kaggle.com/code/rockystats/bbc-text-classification-word2vec-vs-tf-idf/data`

[4] Link for train_test_split method: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

to number using *LabelEncoder library* [5]. In both datasets there aren't missing value but there are 1185 and 99 duplicates respectively on the first and second one.

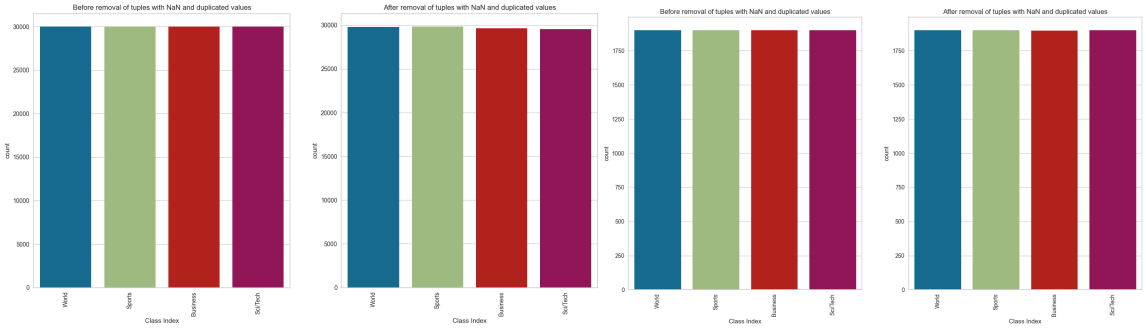
[5]Link for LabelEncoder library: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html`

# 2 — Preprocessing

Before building models, is necessary preprocess dataset in order to remove first of all missing values, duplicates and then clean the text. The entire process is shown in this chapter.

## 2.1 Removing Duplicates and Missing Values

The first thing is remove duplicate rows and those contain missing values. In the images below is possible to see that after removing values the new datasets are balanced as originals and the number of elements of each class are quiet the same that previous.




**(a)** Training Set **(b)** Test Set

**Figure 1:** Class Distribution AG News

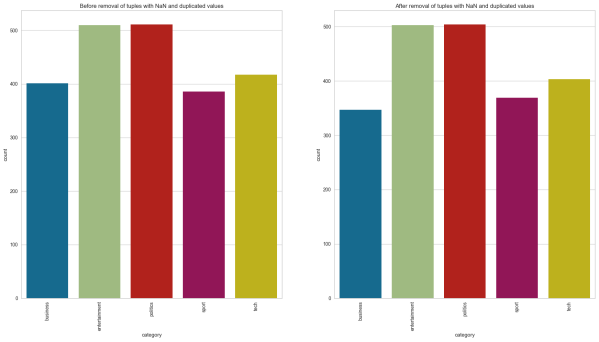



**Figure 2:** Class Distribution BBC News

## 2.2 Text Preprocess

In order to perform computational tasks on text, is need to convert the language of text into a language that the machine can understand. In particular text cleaning is composed by following steps:

- Normalization
- Stemming
- Lemmatization

### 2.2.1 Normalization

One of the key steps in processing language data is to remove noise so that the machine can more easily detect the patterns in the data. Text data contains a lot of noise, this takes the form of

special characters (such as URLs, HTML tags, diacritics, extra white spaces), punctuation and numbers. Additionally, it is also important to apply some attention, if text includes both upper case and lower case versions of the same words then the computer will see these as different entities. To avoid this problem is enough transform all words in text to lowercase. The python library used to implement this steps is *texthero*[6]. Another important phase is removing stop-words, it is list of generic words for example *'i','you','a','the','he','which' etc.* for the English vocabulary. The list of stop-words used is the default in *nltk library*[7]. There are another feature only available for English text, is to write abbreviations in their long forms and slangs in to the correct form, using *contractions library*[8].

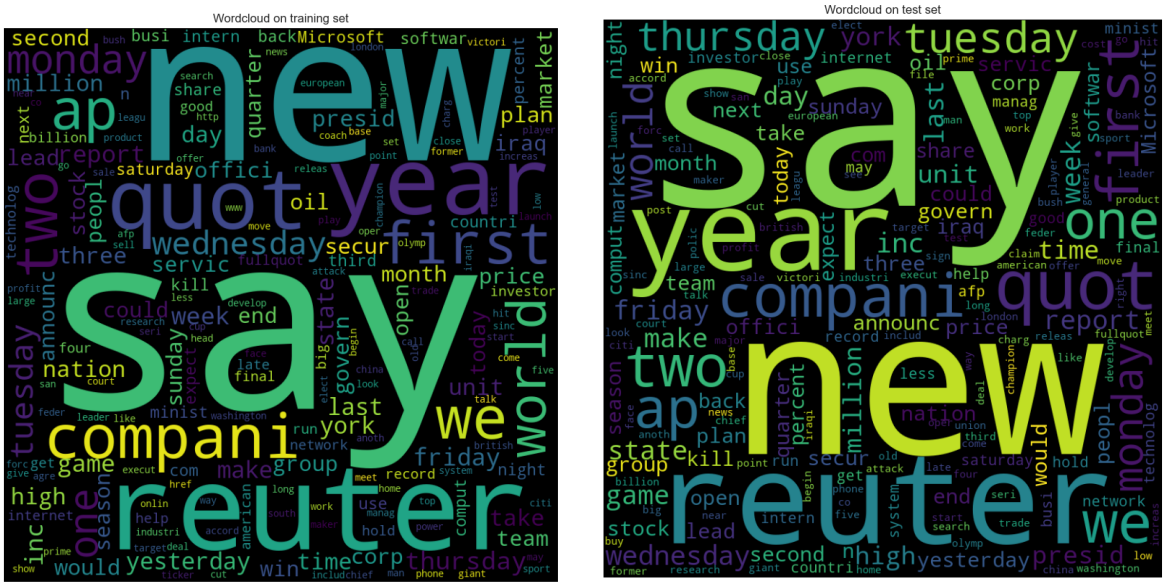
### 2.2.2 Stemming

Stemming is the process of reducing words to their root form. For example, the words *'rain', 'raining' and 'rained'* have very similar, and in many cases, the same meaning. The process of stemming will reduce these to the root form of *'rain'*. This is a way to reduce noise and the dimensionality of data. To implement this process is used *texthero library*, used in the previous step.

### 2.2.3 Lemmatization

The goal of lemmatization is the same as for stemming, in that it aims to reduce words to their root form. However, stemming is known to be a fairly crude method of doing this. Lemmatization, on the other hand, is a tool that performs full morphological analysis to more accurately find the root. To implement this*simplemma library*[9].

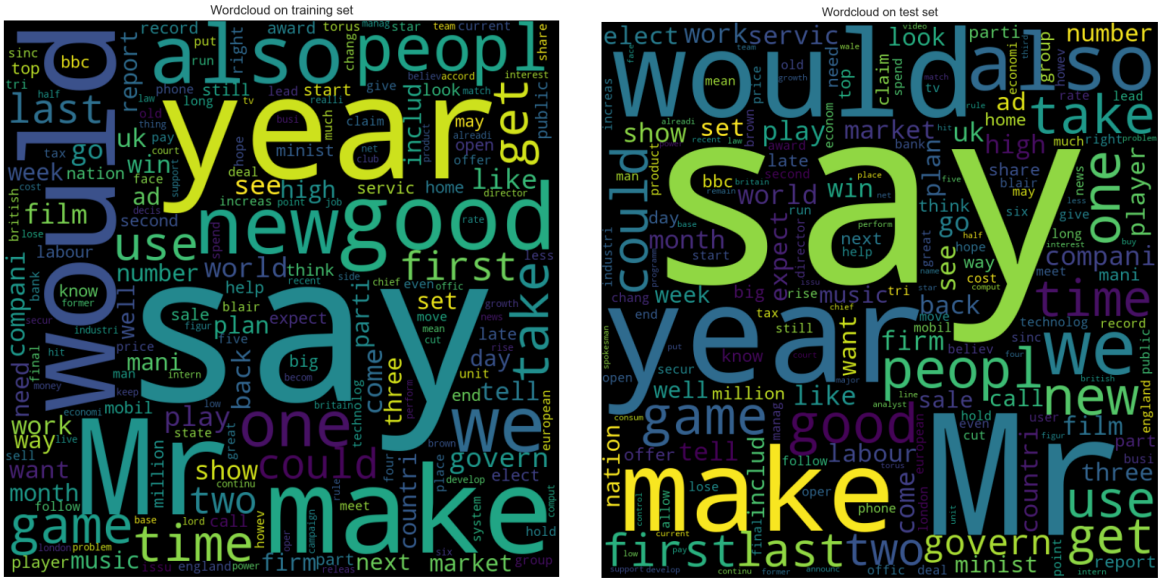
After apply this process to the original datasets these are the results:




**(a)** Training Set **(b)** Test Set

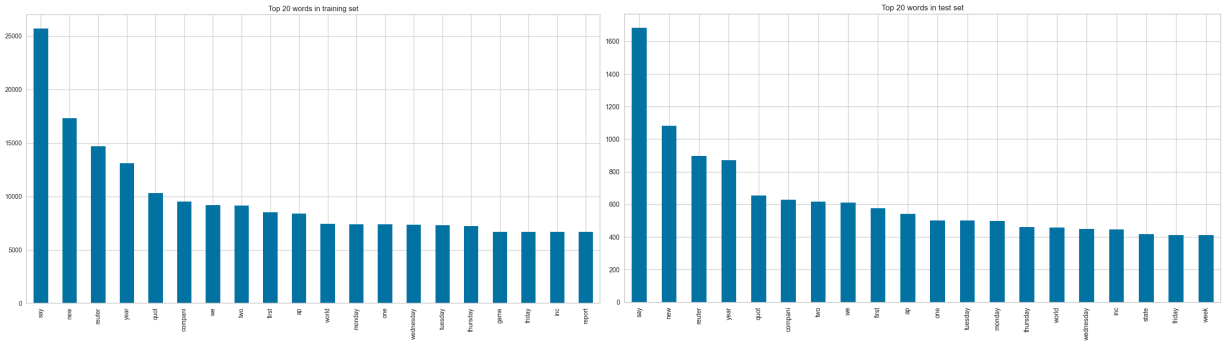
**Figure 3:** Wordcloud AG News

[6]Link for Texthero library: `https://texthero.org`
[7]Link for Nltk library: `https://www.nltk.org`
[8]Link for Contractions library: `https://libraries.io/pypi/contractions/0.1.73`
[9]Link for Simplemma library: `https://libraries.io/pypi/simplemma`



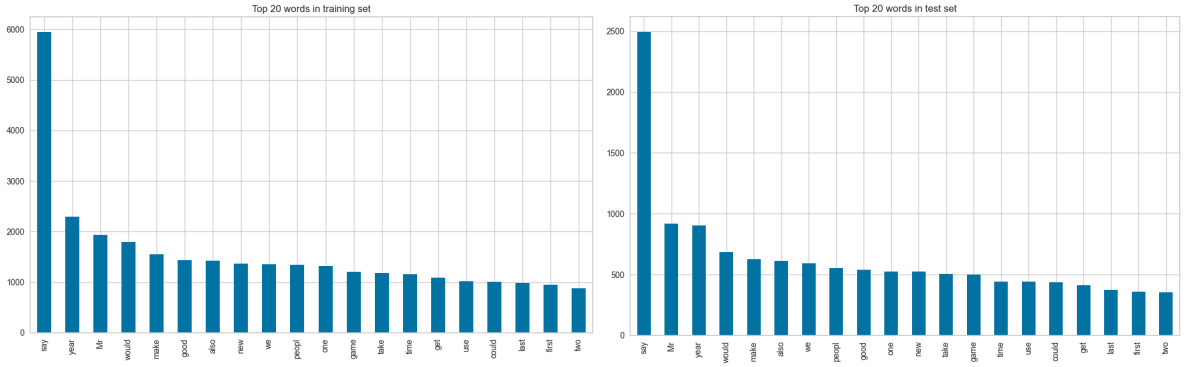

**(a)** Training Set **(b)** Test Set

**Figure 4:** Wordcloud BBC News



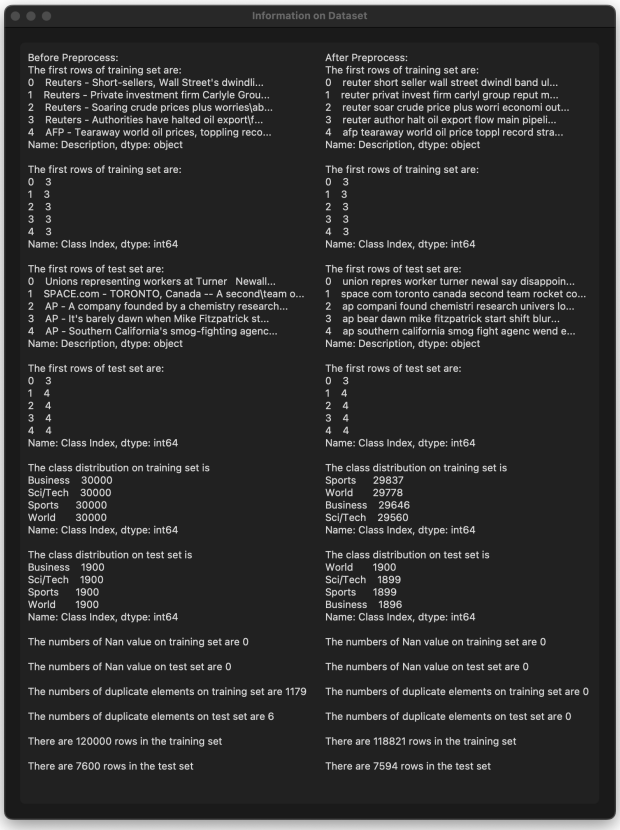

**(a)** Training Set **(b)** Test Set

**Figure 5:** Top 20 Words on AG News




**(a)** Training Set **(b)** Test Set

**Figure 6:** Top 20 Words on BBC News




**(a)** AG News


**(b)** BBC News

**Figure 7:** Before and After Preprocess

# 3 — Classification

After the preprocessing phase, the dataset is ready to be used to learn classification models that will be used in the final application. In this chapter are discussed the chosen strategies relative to classification phase. Before apply classification is necessary to transform text into vector of real numbers, which is the format that ML models support. The process to convert text data into numerical data/vector, is called vectorization.

## 3.1 Vectorization

The solution used to implement vectorization is **Term Frequency-Inverse Document Frequencies** *(TF-IDF)* [10]. It is a numerical statistic that's intended to reflect how important a word is to a document. Words that get repeated too often don't overpower less frequent but important words. It is composed by two parts:

1. **Term Frequency** *(TF)*. It can be understood as a normalized frequency score and it is always $\leq 1$. It is calculated via the following formula:

$$TF = \frac{Frequency\,of\,word\,in\,a\,document}{Total\,number\,of\,words\,in\,that\,document}$$

2. **Inverse Document Frequency** *(IDF)*, but before is necessary make sense of *DF* – *Document Frequency*. It's given by the following formula:

$$DF\,(word) = \frac{Number\,of\,documents\,with\,word\,in\,it}{Total\,number\,of\,documents}$$

   It measures the proportion of documents that contain a certain word. *IDF* is the reciprocal of the Document Frequency, and the final IDF score comes out of the following formula:

$$IDF\,(word) = \log\left(\frac{Total\,number\,of\,documents}{Number\,of\,documents\,with\,word\,in\,it}\right)$$

   The intuition behind it is that the more common a word is across all documents, the lesser its importance is for the current document. A logarithm is taken to dampen the effect of IDF in the final calculation. The final *TF-IDF* score comes out to be:

$$TF\,-IDF\,= TF\,\cdot IDF$$

The higher the score and more important that word is. Basically, the value of a word increases proportionally to count in the document, but it is inversely proportional to the frequency of the word in the corpus.

## 3.2 Classifiers

It's time to train a machine learning models on the vectorized dataset. To minimize lengthy re-training and allow you to share, commit, and re-load pre-trained machine learning models is used *Dill library* [11], that is a useful Python tool that allows to save ML models. Every times that application starts, it controls if models are present in *mode_saved* directory. If a model is present, it is being loaded otherwise it is being trained and saved in *mode_saved* folder.

[10] Link for TfidfVectorizer library: `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

[11] Link for Dill library: `hhttps://libraries.io/pypi/dill`

Another important things is that, every times that the user change a setting (for example changing dataset), all models are primarily removed, then trained and at the end saved.

To find the optimal parameters from the chosen classifiers is performed a technique called *GridSearchCV* [12]. The performance of a model significantly depends on the value of hyperparameters. There is no way to know in advance the best values for hyperparameters so ideally, is necessary to try all possible values to know the optimal values. Doing this manually could take a considerable amount of time and resources and thus a solution is use GridSearchCV to automate the tuning of hyperparameters.

In this application the chosen classifiers are:

- *MultinomialNB*[13]
- *Logistic Regression*[14]
- *SGD Classifier* [15]

### 3.2.1 Hyperparameters

This section is focused on GridSearchCV and hyperparameters of its classifiers. The parameters of the estimator used are optimized by cross-validated using *StratifiedKFold*[16] over a parameter grid. The parameters grid for each classifier is:

- *MultinomialNB*
  - alpha: (1, 0.8, 0.7, 0.5, 0.3, 0.1, 0.05, 0.01, 0.001, 0.0001, 0.00001)
- *Logistic Regression*
  - C : [25, 20, 15, 10, 5, 3, 1, 0.1, 0.05, 0.01]
  - solver: [$'liblinear'$, $'newton-cg'$]
- *SGD Classifier*
  - eta0: [0.0, 0.03, 0.01, 0.003, 0.001, 0.0003],
  - penalty: [$'l1'$, $'l2'$, $'elasticnet'$]
  - alpha: [1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001]
  - loss: [$'log\_loss'$, $'modified\_huber'$]

Here are reported the best parameters, best score and time to the fit classifiers for each dataset:

[12]Link for GridSearchCV library: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

[13]Link for MultinomialNB library: `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html`

[14]Link for Logistic Regression library: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

[15]Link for SGDClassifier library: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html`

[16]Link for StratifiedKFold library: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html`

**AG News**

- *MultinomialNB*
  - best parameters
    - alpha: 0.3
  - best score: 0.895
  - time: 35.087 seconds
- *Logistic Regression*
  - best parameters
    - C: 3
    - solver: 'liblinear'
  - best score: 0.9
  - time: 402.227 seconds
- *SGD Classifier*
  - best parameters
    - eta0: 0.0
    - penalty: 'l2'
    - alpha: 0.0001
    - loss: 'modified_huber'
  - best score: 0.901
  - time: 1121.933 seconds

**BBC News**

- *MultinomialNB*
  - best parameters
    - alpha: 0.7
  - best score: 0.977
  - time: 4.358 seconds
- *Logistic Regression*
  - best parameters
    - C: 3
    - solver: 'newton-cg'
  - best score: 0.979
  - time: 10.081 seconds
- *SGD Classifier*
  - best parameters
    - eta0: 0.0
    - penalty: 'l2'
    - alpha: 0.0003
    - loss: 'modified_huber'
  - best score: 0.979
  - time: 53.516 seconds

The time to fit all classifiers is calculated using *Apple M1* processor.

# 4 — Results

In this chapter there is the description of the results obtained using following statistics:

- *Classification Report*[17]
- *Confusion Matrix*[18]
- *Area under the Curve*[19]
- *Class Prediction Error*[20]

The following sections report results obtained for each dataset.

## 4.1 AG News

### 4.1.1 Classification Report

| MultinomialNB | | | | |
|---|---|---|---|---|
| | Precision | Recall | F1-Score | Support |
| World | 0.91 | 0.89 | 0.90 | 1900 |
| Sports | 0.94 | 0.97 | 0.96 | 1899 |
| Business | 0.87 | 0.84 | 0.85 | 1896 |
| Sci/Tech | 0.85 | 0.88 | 0.87 | 1899 |
| Accuracy | | | 0.90 | 7594 |
| Macro Avg | 0.89 | 0.90 | 0.89 | 7594 |
| Weighted Avg | 0.89 | 0.90 | 0.89 | 7594 |

Final Training Accuracy: 91% Model Accuracy: 90%

| Logistic Regression | | | | |
|---|---|---|---|---|
| | Precision | Recall | F1-Score | Support |
| World | 0.93 | 0.90 | 0.91 | 1900 |
| Sports | 0.95 | 0.98 | 0.96 | 1899 |
| Business | 0.88 | 0.87 | 0.88 | 1896 |
| Sci/Tech | 0.89 | 0.89 | 0.89 | 1899 |
| Accuracy | | | 0.91 | 7594 |
| Macro Avg | 0.91 | 0.91 | 0.91 | 7594 |
| Weighted Avg | 0.91 | 0.91 | 0.91 | 7594 |

Final Training Accuracy: 95% Model Accuracy: 91%

[17]Link for Classification Report library: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html`

[18]Link for ConfusionMatrixDisplay library: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html`

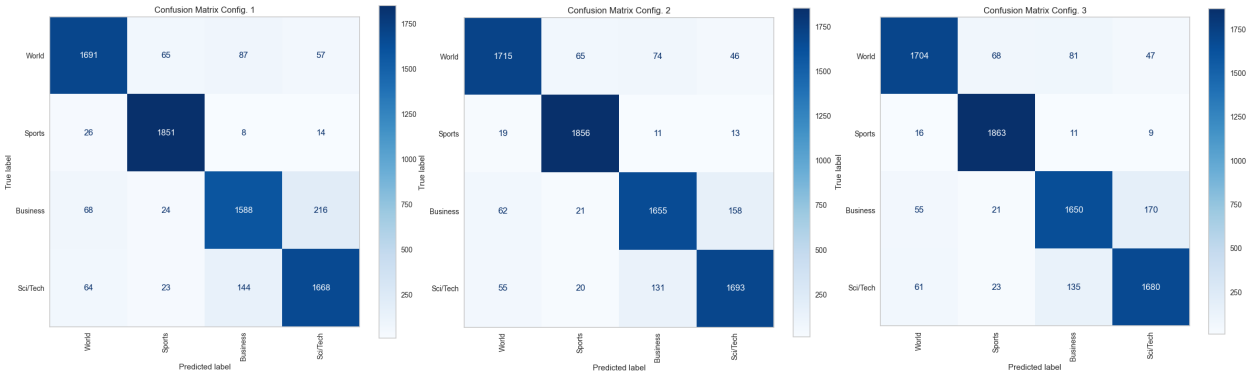
[19]Link for ROCAUC library: `https://www.scikit-yb.org/en/latest/api/classifier/rocauc.html`

[20]Link for ClassPredictionError library: `https://www.scikit-yb.org/en/latest/api/classifier/class_prediction_error.html`

| SGD Classifier | | | | |
|---|---|---|---|---|
| | Precision | Recall | F1-Score | Support |
| World | 0.93 | 0.90 | 0.91 | 1900 |
| Sports | 0.94 | 0.98 | 0.96 | 1899 |
| Business | 0.88 | 0.87 | 0.87 | 1896 |
| Sci/Tech | 0.88 | 0.88 | 0.88 | 1899 |
| Accuracy | | | 0.91 | 7594 |
| Macro Avg | 0.91 | 0.91 | 0.91 | 7594 |
| Weighted Avg | 0.91 | 0.91 | 0.91 | 7594 |

Final Training Accuracy: 95% Model Accuracy: 91%

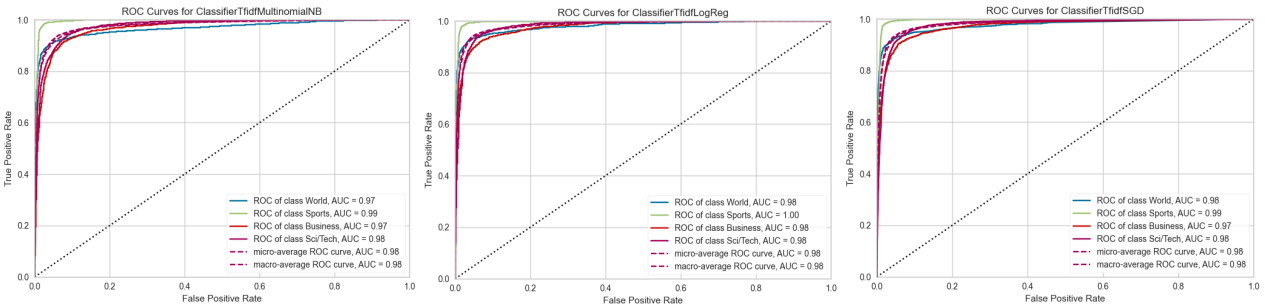
### 4.1.2 Confusion Matrix




**(a)** MultinomialNB **(b)** Logistic Regression **(c)** SGD Classifier

**Figure 8:** Classification Report AG News

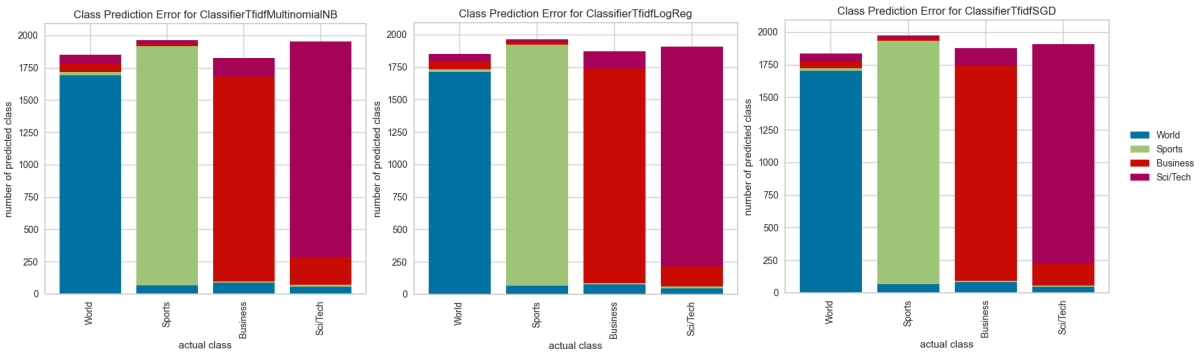
### 4.1.3 Area Under the Curve




**(a)** MultinomialNB **(b)** Logistic Regression **(c)** SGD Classifier

**Figure 9:** Area Under the Curve AG News

### 4.1.4 Class Prediction Error




**(a)** MultinomialNB **(b)** Logistic Regression **(c)** SGD Classifier

**Figure 10:** Class Prediction Error AG News

## 4.2 BBC News

### 4.2.1 Classification Report

| MultinomialNB | | | | |
|---|---|---|---|---|
| | Precision | Recall | F1-Score | Support |
| business | 0.96 | 0.98 | 0.97 | 162 |
| entertainment | 0.99 | 0.92 | 0.95 | 96 |
| politics | 0.95 | 0.98 | 0.97 | 128 |
| sport | 0.99 | 1.00 | 1.00 | 141 |
| tech | 0.95 | 0.95 | 0.95 | 111 |
| Accuracy | | | 0.97 | 638 |
| Macro Avg | 0.97 | 0.97 | 0.97 | 638 |
| Weighted Avg | 0.97 | 0.97 | 0.97 | 638 |

Final Training Accuracy: 99% Model Accuracy: 97%

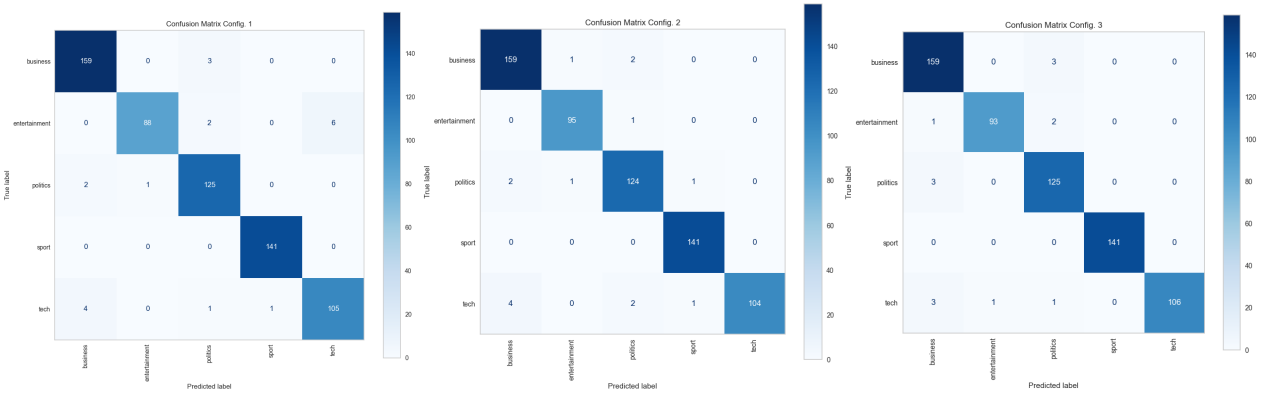
| Logistic Regression | | | | |
|---|---|---|---|---|
| | Precision | Recall | F1-Score | Support |
| business | 0.96 | 0.98 | 0.97 | 162 |
| entertainment | 0.98 | 0.99 | 0.98 | 96 |
| politics | 0.96 | 0.97 | 0.96 | 128 |
| sport | 0.99 | 1.00 | 0.99 | 141 |
| tech | 1.00 | 0.94 | 0.97 | 111 |
| Accuracy | | | 0.98 | 638 |
| Macro Avg | 0.98 | 0.98 | 0.98 | 638 |
| Weighted Avg | 0.98 | 0.98 | 0.98 | 638 |

Final Training Accuracy: 100% Model Accuracy: 98%

| SGD Classifier | | | | |
|---|---|---|---|---|
| | Precision | Recall | F1-Score | Support |
| business | 0.98 | 0.98 | 0.98 | 162 |
| entertainment | 0.99 | 0.99 | 0.99 | 96 |
| politics | 0.96 | 0.98 | 0.97 | 128 |
| sport | 0.99 | 1.00 | 1.00 | 141 |
| tech | 1.00 | 0.96 | 0.98 | 111 |
| Accuracy | | | 0.98 | 638 |
| Macro Avg | 0.99 | 0.98 | 0.98 | 638 |
| Weighted Avg | 0.98 | 0.98 | 0.98 | 638 |

Final Training Accuracy: 100% Model Accuracy: 98%

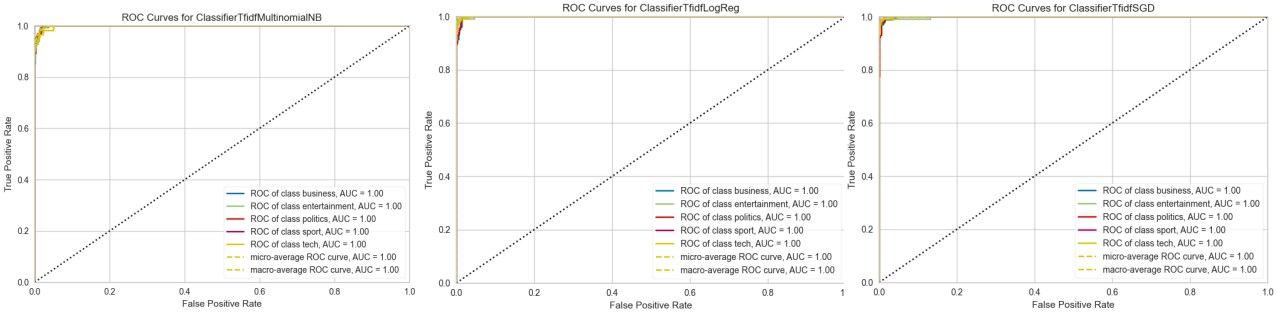
### 4.2.2 Confusion Matrix




**(a)** MultinomialNB **(b)** Logistic Regression **(c)** SGD Classifier

**Figure 11:** Classification Report BBC News

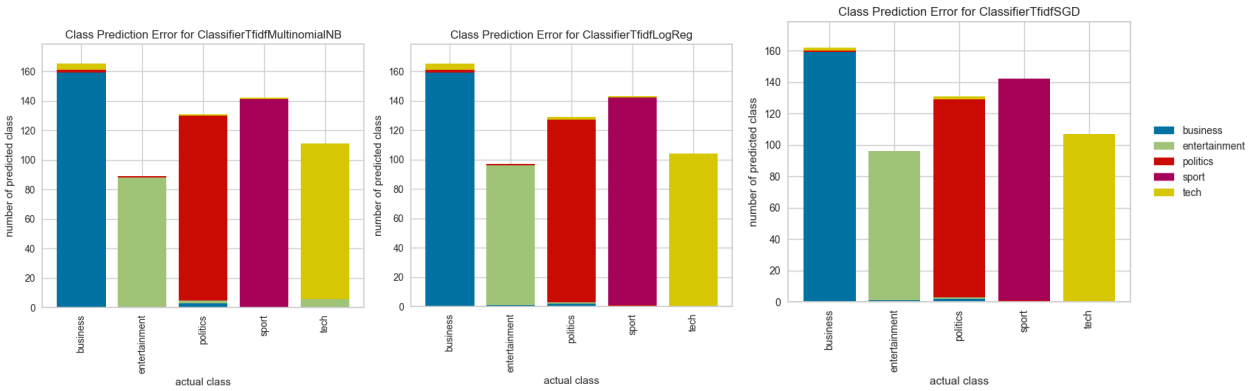
### 4.2.3 Area Under the Curve




**(a)** MultinomialNB **(b)** Logistic Regression **(c)** SGD Classifier

**Figure 12:** Area Under the Curve BBC News

### 4.2.4 Class Prediction Error




**(a)** MultinomialNB **(b)** Logistic Regression **(c)** SGD Classifier

**Figure 13:** Class Prediction Error BBC News

# 5 — Application

**News Classifier** is a simple application based on text mining in order to classify news articles to their right topic.

## 5.1 How to Install

Before use the application is necessary to install all library used using this command: *pip install -r requirements.txt*. After installing all libraries, the command to start application is *python ./src/main.py*. The application is tested with *python 3.8*.

## 5.2 How to classify text

Every user can operate with the applicative as they open it and insert into the textbox a news to classify.

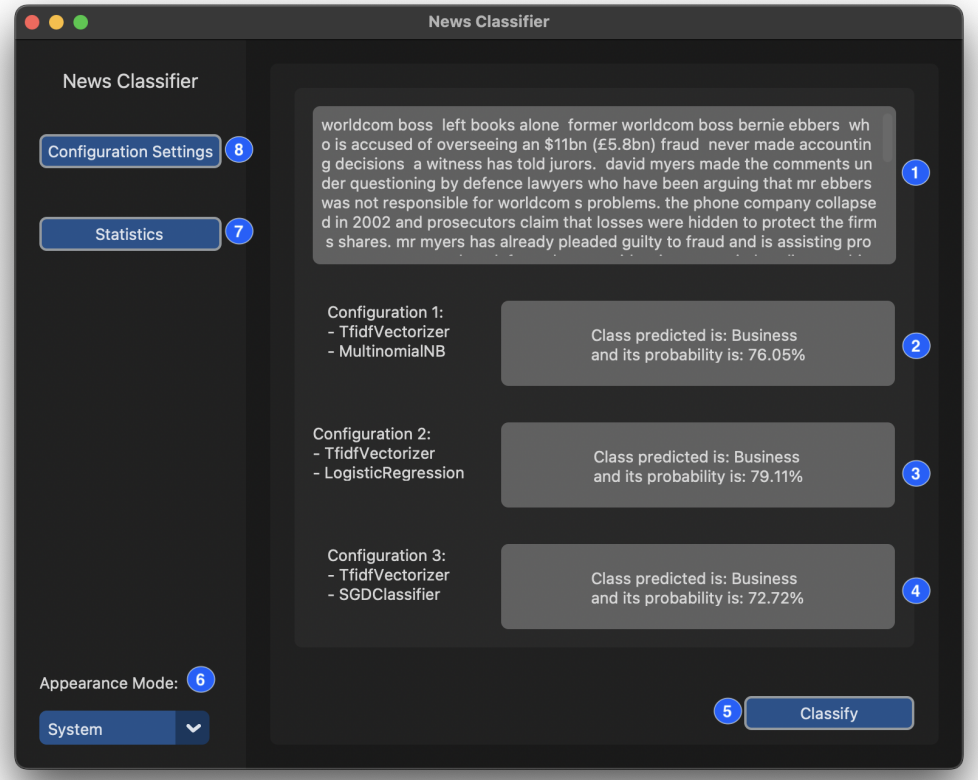



**Figure 14:** Home Application

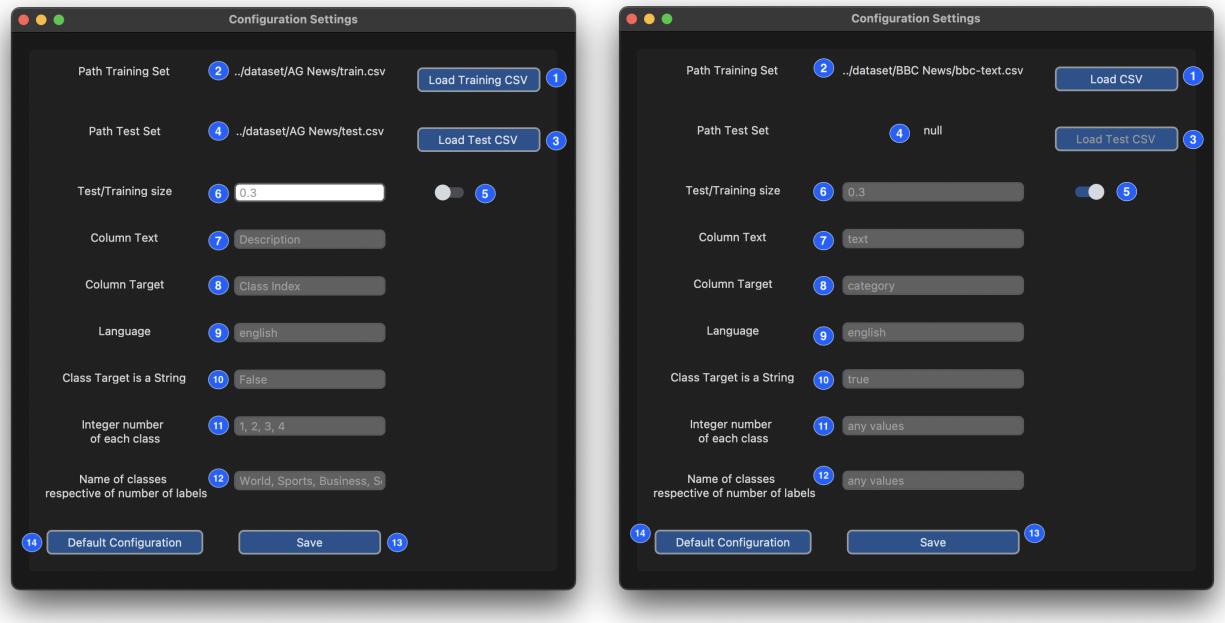
1. *TextBox*: Input text to classify
2. *Prediction first classifier*: Output text displays class predicted, by the first classifier, and its probability
3. *Prediction second classifier*: Output text displays class predicted, by the second classifier, and its probability

4. *Prediction third classifier*: Output text displays class predicted,by the third classifier, and its probability
5. *Classify*: Button to classify text inserted in *TextBox*
6. *Appearance*: Option menu to change the appearance of the application (Dark, Light and System)
7. *Statistics*: Button to see all statistics on dataset and classifiers
8. *Configuration Settings*: Button to change dataset

### 5.3 How to change dataset

If the user wants to change dataset, the steps are:

- Select one or two *.csv files* as input
- Insert the configuration values for relative dataset given in input




**(a)** Configuration AG News **(b)** Configuration BBC News

**Figure 15:** Configuration

1. *Load Training CSV/Load CSV*: Button to import training or dataset file
2. *Path Training Set*: Text to display the path of training file
3. *Load Test CSV*: Button to import test file (if switch is on)
4. *Path Test Set*: Text to display the path of test file or null if the dataset is composed by one file
5. *Switch*: Status on to load dataset composed by only one file and set *Test/Training Size* value. Status off load trainig and test files

6. *Test/Training Size*: Value of the test size (only if there is only one file)
7. *Column Text*: CSV column name to detect text to classify
8. *Column Target*: CSV column name contains classes
9. *Language*: Text to set language of the text to classify
10. *Class Target is a String*: Boolean value: true if classes are described by strings, false by number
11. *Integer Number of Each Class*: Text to set the list of numeric classes when *Class Target is a String* is false
12. *Name of classes respective of number of label*: Text to set the list of classes name when *Class Target is a String* is false
13. *Save*: Button to save settings
14. *Default Configuration*: Button to restore default settings

## 5.4 Display statistics

The image below describes all statistics in the application, each buttons display the corresponding statistics.

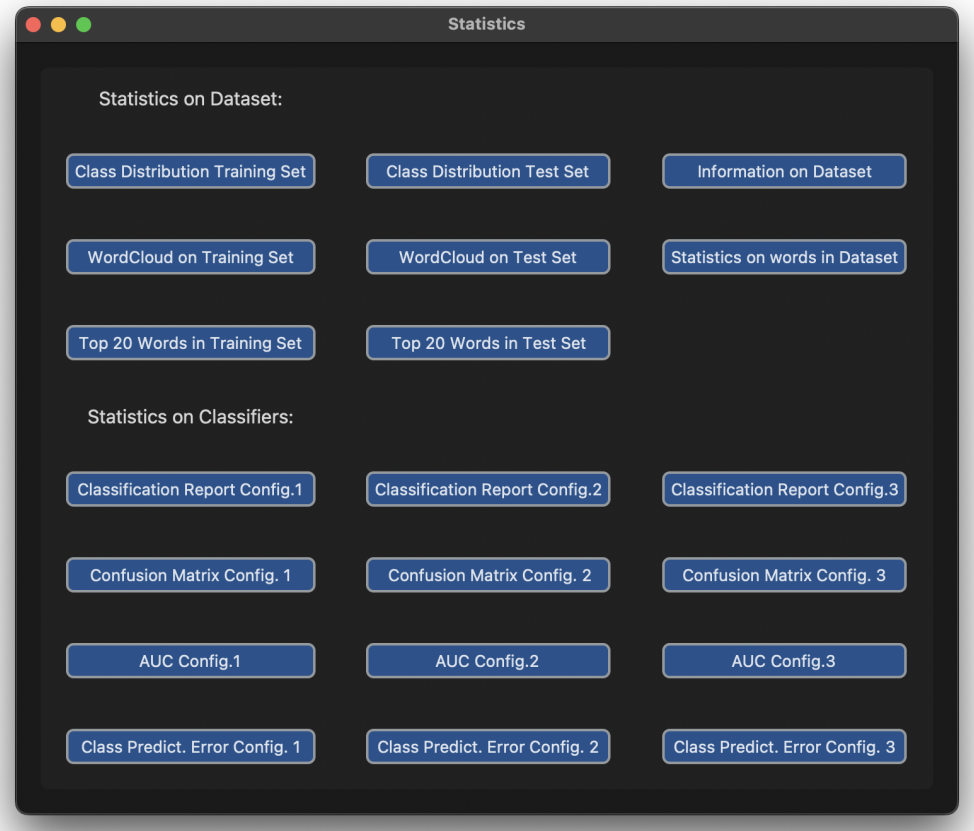



**Figure 16:** Statistics