Hogeschool van Amsterdam

ONBOARDING ASSIGMENT

MACHINE LEARNING REPORT

BONNAVAUD Laura

February 2023

TABLE OF CONTENTS

I. Introduction

a. Problem definition

Machine learning is the process whereby computers learn to make decisions from data without being explicitly programmed.

For this onboarding assignment, I need to work on sentiment analysis of tweet data.

The task is usually modeled as a classification problem where using some historical data with known sentiment, we need to predict the sentiment of a new piece of tweet.

The aim is to provide an overview of the different aspects of a typical data science project, from data collection to model building and visualization.

b. Development environment

I code the machine learning algorithms in the **python** programming language, which is an interpreted programming language. I will also use **Jupyter**, which is a software that offers to create "notebooks". It allows me to easily keep track and debug parts of code.

The machine learning libraries used in the project to simplify the code while making it more efficient are mainly the following:

- **Numpy:** used for manipulating N-dimensional matrices and performing operations on it.
- **Pandas:** used for manipulating two-dimensional arrays which defines the DataFrame type in order to manipulate our CSV file data.
- **Matplotlib:** used for drawing different graphs from data.
- **Scikit-learn:** used for performing the data pre-processing and model creation steps.

II. Overview of the proposed solution

a. Solution steps

To address this problem, I will follow a few steps :

- Get the data
- Preprocessing: I need to clean and preprocess the data before building the model
- Preparation: the data must be ready to use
- Visualization: visualize the data with a word cloud and charts
- Model building: once the features are extracted, I can use machine learning algorithms such as Naive Bayes, Support Vector Machines (SVM), or Neural Networks to train the model on the training dataset
- Model Evaluation: after training the model, I need to evaluate its performance on the testing dataset. I can use metrics such as accuracy, precision, recall, and F1-score to measure the performance of the model.

b. Type of machine learning system

There are so many different types of machine learning systems:

- depending on whether the learning takes place under human supervision (supervised, unsupervised, semi-supervised or reinforced learning)

- depending on whether learning takes place gradually or not, gradually (e-learning or group learning)
- depending on whether it is content to compare the new data with data known, or that it detects, on the contrary, structuring elements in the training data and builds a predictive model like a scientific (learning from observations or learning from a model)

Supervised learning is a type of machine learning where the values to be predicted are already known, and a model is built with the aim of accurately predicting values of previously unseen data. Supervised learning uses features to predict the value of a target variable. There are two types:

- Classification: is used to predict the label, or category, of an observation.
- Regression: is used to predict continuous values.

Thus, knowing the sentiment of a tweet can be learned step by step by based on a deep neural network model which is trained on examples of positive or negative tweet: this makes it a supervised system learning from a model. In our case, the positive feeling is associated with 4 and negative with 0.

c. Select a performance measure

An important thing that I learned during my research is that the evaluation of machine learning models does not depend on the chosen algorithms, but is done on the whole validation when choosing the hyperparameters and/or the model, on the test set at the end of the process and on the training set.

Once the model has been created, it is possible to evaluate the model. For this, many indicators can be used, the majority depending on the task chosen. In this project, I will use this indicators:

- **Confusion Matrix:** In the classification task, the confusion matrix is the main indicator of model quality. It is a double-entry table matching the actual classes and the classes predicted by the model. In the matrix, there are 4 cells: true positives (VP), true negatives (VN), false positives (FP), false negatives (FN).
- **Accuracy:** Accuracy is the first indicator deviated from this matrix. It consists of looking at which portion of the predictions is correct. Its formula is therefore for a binary classification: accuracy=(VP+VN)/total. The goal is to classify tweets as positive or negative with at least 80% accuracy, so the minimum performance required would be 80% accuracy.

Accuracy, which does not distinguish the type of error made, is supplemented by two other indicators: recall and precision:

- **Recall:** it is the proportion of positive class detected. A strong recall therefore indicates that almost all cases in the positive class have been detected.
- **Precision:** it is the proportion of true positives in all positives detected. This makes it possible to estimate the number of positive tweet.
- **F1-score:** it is based on precision and recall; it gives a unique indicator.

d. Selection of classification models

There are many classifications models that I can use, I must find the most relevant to the problem.

I make a list of the main models that seem important to me and I explain theoretically what could be good or bad for the sentiment analysis:

- **Random Forests:**

Random Forest are capable of handling non-linear relationships between features and target variables, and can also handle interactions between features. Additionally, by creating multiple decision trees on subsets of data, random forests can reduce overfitting and improve generalization performance.

However, one potential disadvantage of random forests is that they can be computationally expensive, especially when dealing with large datasets with many features.

- **K-Nearest Neighbors:**

KNN is a method that does not create a model as such: the set of training data constitutes the model.

However, it may not be the best choice for sentiment analysis because it is sensitive to the scaling of the data and may require normalization of the input data. Additionally, it can be computationally expensive and may not work well with high-dimensional data.

- **Logistic Regression:**

Linear regression consists of the creation of a linear function of the variables which associates positive numbers with the data of the positive class and negative values with the negative class and the application of the function on the result obtained. This model is easy to implement, and it is a linear model that can be easily trained and optimized. It is often used in combination with feature extraction techniques, such as bag-of-words, to extract meaningful features from text data.

- **Naive Bayes:**

Naive Bayes is a probabilistic classifier that assumes that the presence of a particular feature is independent of the presence of other features. This classification uses the conditional probabilities P(B|A) (Bayes theorem) and makes a "naive" assumption. This model may be good because it is simple, fast, and can handle high-dimensional data.

- **Support Vector Machine:**

It is a set of techniques generalizing linear classifiers. There are advantages to using this model: it is not limited by the number of dimensions. SVMs can be optimized to find the best decision boundary for the data, which can lead to high accuracy. However, SVMs can be computationally expensive and may require feature selection to avoid overfitting.

- **Support Vector Classifier:**

It is a type of supervised learning algorithm that can be used for binary classification or multi-class classification problems. The objective of SVC is to find a hyperplane in a high-dimensional space that separates the classes in the best way possible. It can work well for sentiment analysis.

III. Data understanding

a. Import the dataset

The dataset use for this onboarding assignment are the dataset from Sentiment140, (http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip) and a handmade test set of at least 10 tweets.

b. Data analysis

First of all, I make a copy of the data. I'm creating a Jupyter notebook to keep track of my data exploration.

Let's study the datasets: firs, I load the data from the SVC file by using the Pandas read_csv function.

```
#Load the data from the CSV file
columns=["sentiment", "id", "date", "query", "user", "text"]

df_test = pd.read_csv("testdata.manual.2009.06.14.csv", header=None, names=columns, encoding="ISO-8859-1")
df_train = pd.read_csv("training.1600000.processed.noemoticon.csv",  header=None, names=columns, encoding="ISO-8859-1")
```

A first problem occurred during the simple reading of the file. The CSV file provided by Sentiment140 does not have a header row, and the default encoding used by Pandas (which is UTF-8) may not be compatible with the file. So I need to add the arguments : **header=None**, **names=columns**, and **encoding="ISO-8859-1" to the function.**

- **header=None**: this tells Pandas that the CSV file does not have a header row. Without this argument, Pandas assumes that the first row of the CSV file contains column names and uses them as the column names in the resulting DataFrame.
- **names=columns**: this tells Pandas to use the column names provided in the **columns** variable as the column names in the resulting DataFrame.
- **encoding="ISO-8859-1"**: this specifies the character encoding used by the CSV file. The Sentiment140 dataset uses ISO-8859-1 encoding, which is different from the default UTF-8 encoding used by Pandas.

Now, let's study the characteristics of each variable: I can examinate the first five lines using the head() method of the training dataset:

```
print(df.head())
```

```
   sentiment          id                          date     query  \
0          0  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY
1          0  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY
2          0  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY
3          0  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY
4          0  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY

              user                                                text
0  _TheSpecialOne_  @switchfoot http://twitpic.com/2y1zl - Awww, t...
1    scotthamilton  is upset that he can't update his Facebook by ...
2         mattycus  @Kenichan I dived many times for the ball. Man...
3          ElleCTF    my whole body feels itchy and like its on fire
4           Karoli  @nationwideclass no, it's not behaving at all....
```

First five lines of the data set

The info() method provides a description of the data, in particular the total number of rows, the type of each variable and the number of non-zero values:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   sentiment  1600000 non-null  int64
 1   id         1600000 non-null  int64
 2   date       1600000 non-null  object
 3   query      1600000 non-null  object
 4   user       1600000 non-null  object
 5   text       1600000 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

The dataset has 6 columns, the first which indicates the number of the associated sentiment, the second the identifier, the third the date on which the tweet was posted, the fourth query, the fifth the name of the user who wrote the tweet and last the text of the tweet.

There are 1,600,000 rows in this table. We see in the non-zero column that no variable has districts with missing data.

Now, let's look at the details of the number of positive and negative sentiment in the training and test dataset:

```
df_train.sentiment.value_counts()
```

```
Out[1]: 0    800000
        4    800000
        Name: sentiment, dtype: int64
```

In the training dataset, we can see there is as many positive as negative sentiments:

```
df_test.sentiment.value_counts()
```

```
Out[2]: 4    182
        0    177
        2    139
        Name: sentiment, dtype: int64
```

For the test dataset, we can see there is 182 positive sentiments and 177 negative sentiments. I will not use the neutral sentiments by choice.

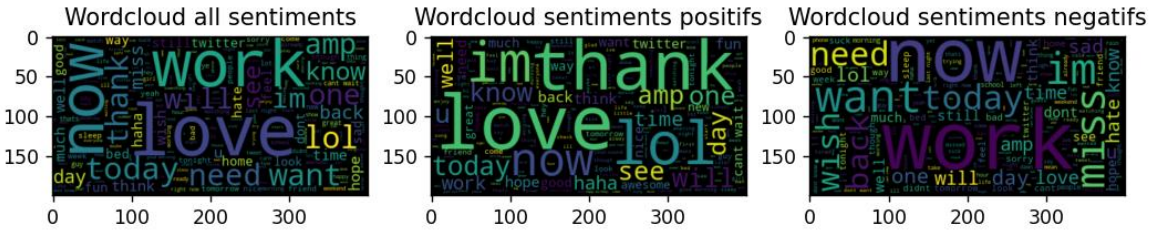
c. Visualization

I visualize the data of the training dataset with some word clouds and charts:

- **Word clouds:**

I can create 3 word clouds to see the most frequently used words: one for all the data, one for just the positive sentiments and one for the negative. A word cloud is an image composed of words with different sizes and colors. The size of the text corresponds to the frequency of the word. The more frequent a word is, the bigger and bolder it will appear on the word cloud.

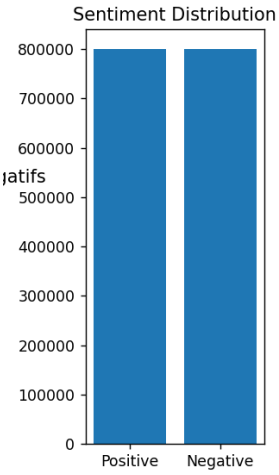
I use the Word Cloud function from the word cloud package and import matplotlib.pyplot which will allow word cloud to plot on its base:

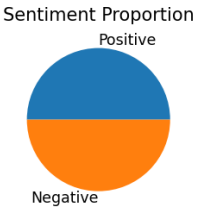


During this step, I had a stupid problem when creating word clouds. To avoid having to recompile all my code, I wanted to code my word clouds in another file. However, I named it “wordcloud.py”. Returning to my main code, I had this error: "*ImportError: cannot import name 'WordCloud' from partially initialized module 'wordcloud' (most likely due to a circular import)*". I searched for a long time, even after checking the library installation, and finally it was the result of conflict with my local file name and python library name.

- **Charts:** bar charts & pie charts

I created a bar chart and a pie chart to visualize the distribution of positive and negative sentiments in the training dataset:





For display all this word clouds and charts on the same window, I use subplots in matplotlib.

d. Handmade test set

Now that I know the details of the two datasets, I need to create the handmade test set.

There are three possibilities for this creation:

- create a .csv file with the same characteristics as the two we have, i.e. with the same number of columns
- create a .csv file with only the two columns that interest us: the sentiment and text column
- code the tweets within the code itself

To start, I choose to create a .csv file with the two columns that interest me, which seems to me the simplest. I filled the csv file with for the positive tweet positive known expressions and for the negative tweets random sentences with a negative meaning.

To create this csv file, I entered the data in an excel file, in the first column the number of the associated sentiment and in the second the text, I then saved this file in csv format.

But I had a few error: firs, I had this error : *"AttributeError: Can only use .str accessor with string values!".* To fix this, I try to force the values in the "text" column to be strings by using the **astype()** method.

But then, I realize when reading the csv file with the metho **read_csv** that the column wasn't correct: the column sentiment contains some text but not the column text. So, I change and save the file as a CSV file by selecting "*Save As*" and choosing "*CSV (Comma delimited*)" as the file format, and with the format UTF-8. In the "*Save As*" dialog box, I make sure to change the delimiter to a comma (,) instead of the default semicolon (;). But that doesn't change anything. Then, I try the to use the **read_csv** function from pandas library with the correct delimiter ";" to load the CSV file into a data frame.

This problem persisted, so I could not use this handmade set to test my models.

IV. Data cleaning

The data preparation phase makes it possible to move from raw data (as extracted from data sources) to data that can be used by the various Machine Learning algorithms. Here, machine learning algorithms can only work properly with good data.

A problem arises when the text type variables of our tweets. Most machine learning algorithms work on numbers: I need to convert these strings into numbers.

I also need to limit the data used for the rest of the process. All this preparation is done so that the results are as consistent as possible.

a. Delete columns

With pandas, columns can be deleted in two ways, either by specifying the list of columns to keep, or by directly indicating the names of the columns to be deleted with the drop function.

Here, I eliminate unnecessary columns : as we saw in the data analysis part, we have 6 columns, two of which are of interest to us for the rest of the project. I will therefore neglect the others by using .drop function for the two datasets :

```
#drop unnecessary columns
df_train = df_train.drop(["id", "date", "query", "user"], axis=1)
df_test = df_test.drop(["id", "date", "query", "user"], axis=1)
```

For the test dataset, I saw some neutral sentiment. I can predict a problem when using the test dataset with neutral sentiments, because I will not have worked with these neutral sentiments before. By choice, I will neglect neutral feelings because in the training dataset we only have positive and negative. For that, I delete all the neutral sentiment rows in the test dataset.

```
#drop the sentiment neutre in the test df
df_test.drop(df_test[df_test['sentiment'] == 2].index,  inplace = True)
```

b. Limit text and normalization

We don't need all the raw data present in our dataset. To optimize the text so that we only keep the words that are of interest to us: those that make sense to determine whether the sentence is positive or negative, we must remove all the unnecessary characters.

For that, I define a function **clean_text** that removes URLs, mentions, hashtags, non-alphanumeric characters:

```
def clean_text(text):
    text = re.sub(r"http\S+", "", text)#remove URLs
    text = re.sub(r"@[A-Za-z0-9]+", "", text) #Remove mentions (@username)
    text = re.sub(r"#[A-Za-z0-9]+", "", text)  #remove hashtags (#hashtag)
    text = re.sub(r"\d", "", text) #remove digits
    text = re.sub(r"[^\w\s]+", "", text) #remove non alphanumerique like emogies
    text = re.sub(r"[^a-zA-Z\s]+", "", text) #remove non alphabetic
    text = re.sub(r"\b(?:am|is|are|was|were|been|being)\b", "", text) #remove all form of verb be
    text = re.sub(r"\b(?:do|does|did|doing)\b", "", text) #remove all forme of verb do
    text = re.sub(r"\b(?:go|goes|went|going)\b", "", text) #verb go
    text = re.sub(r"\b(?:get|gets|got|getting)\b", "", text) #verb get
    text = re.sub(r"\b(?:make|makes|made|making)\b", "", text) #verb make
    text = re.sub(r"\b\w+ly\b", "", text) #all adverb
    text = re.sub(r"\b(?:ah|oh|eh|uh)\b", "", text) #all interjections
```

At first, I was trying to remove all forms of some verbs and adverbs. But while I was doing this, I realized that what I was trying to do was partly two natural language processing (NLP) techniques to reduce words to their base or root form with the aim of standardizing words and reducing the dimensionality of the vocabulary: Lemmatization and stemming:

- **Stemming:** involves removing the suffix from a word to reduce it to its base form. For example, the stem of "walking", "walked", and "walks" is "walk". This process is relatively simple and fast, but it can also result in words that are not actually words (e.g. "walk" is a valid word, but "walks" and "walked" are not).

- **Lemmatization:** involves reducing a word to its base form (called a lemma) using a dictionary-based approach that considers the part of speech of the word. For example, the lemma of "am", "is", "are", "was", and "were" is "be". This process is more accurate than stemming, but it is also more complex and computationally expensive.

However, I read that it is technically possible to perform stemming and lemmatization at the same time, but it is generally not recommended. This is because stemming and lemmatization are two different approaches to reducing words to their base form, and they have different strengths and weaknesses.

At first, for the simplicity, I choose stemming. But with reflection, lemmatization may be a better choice for accuracy and precision.

So, I first added these two lines to the function to remove stop words, and stems the remaining words.

```
# Remove stop words and stem the words
words = text.split()
words = [stemmer.stem(word) for word in words if word not in stop_words]
text = " ".join(words)
```

But then, I import the **WordNetLemmatizer** from the **nltk.stem** module and Icreated an instance of the lemmatizer called **lemmatizer**. In the **clean_text** function, I replace the stemming step with a lemmatization step. Specifically, I use the **word_tokenize** function from the **nltk** library to split the text into individual words, and then apply the **lemmatizer.lemmatize** method to each word. Finally, I join the lemmatized words back into a string and return it. I can erase the lines where I was trying to remove all forms of some verbs, because it is what lemmatization do.

```
words = word_tokenize(text)
words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
text = " ".join(words)
```

I compared the results of the accuracy of a model to see which method is the best, and the accuracy lost 1.5% with the lemmatization technique. So, I left the stemming technique.

Finally, we apply this function to the "text" column of the data.

For example, here is the test dataset cleaned:

	sentiment	text
0	4	looooooooovvvvvvee kindl dx cool fantast right
1	4	read kindl love lee child good read
2	4	ok first asses fuck rock
3	4	youll love kindl ive mine month never look bac...
4	4	fair enough kindl think perfect

c. Vectorize the text data

A machine learning model cannot work with the text data directly, but rather with numeric features we create from the data. I have to use a method called bag-of-words (BOW) : a bag-of-words approach describes the occurrence of words within a document. The column is a word, and the row represents how many times we have encountered it in the respective review.

For execute a BOW process, I use the **CountVectorizer** from **sklearn.feature_extraction.text.**

```
#vectorize the text data using CountVectorizer
vectorizer = CountVectorizer(stop_words='english', max_features=1000, ngram_range = (1,1), encoding='latin-1')
```

This line of code creates a **CountVectorizer** object that will be used to transform the text data into a numerical format that can be used in machine learning algorithms.

I first had a error of memory : "*Unable to allocate 5.96 GiB for an array with shape (1600000, 500) and data type int64".* This is because my system is running out of memory to handle the large array data created by the **CountVectorizer**. For that, I add the parameters:

- The **stop_words** parameter is set to 'english', which tells the vectorizer to remove common English words that don't provide much meaning to the text (e.g., "the", "and", "is").
- The **max_features** parameter is set to 1000, which limits the maximum number of features that the vectorizer will consider. If I don't put the max features, I have a memory problem.
- The **ngram_range** parameter is set to (1,1), which means that the vectorizer will only consider single words (i.e., unigrams) as features.
- The **encoding** parameter is set to 'latin-1', which specifies the character encoding to use when reading the text data.

```
#apply on text columns of df
X_train = vectorizer.fit_transform(df_train['text'])
X_test=vectorizer.transform(df_test['text'])
```

These two lines of code use the **vectorizer** object created in the previous line to transform the text data in the **text** column of the **df_train** and **df_test** dataframes into numerical format. The **fit_transform()** method is called on the training data, which fits the vectorizer to the data and transforms it at the same time. The **transform()** method is called on the testing data, which transforms it using the vectorizer that was fitted to the training data. The resulting **X_train** and **X_test** variables are sparse matrices containing the transformed text data. A sparse matrix only stores entities that are non-zero, where the rows correspond to the number of rows in the dataset, and the columns to the BOW vocabulary.

d. Separate datasets

We must always have two datasets: one for training to create the model and one for testing it. We have one more test dataset who has been creating by my own.

The proportion of training/test datasets is an 80/20 ratio. Here the datasets have already been created for us in advance.

However, after converting the text to a numeric format using **CountVectorize**, problems may occur: even if I have separate training and testing datasets, I still need to split the data into training and testing sets to ensure that the model learns to generalize well to new, unseen data.

The **CountVectorizer** object is fit to the training data only, and the resulting object is used to transform both the training data and the testing data. This means that the same feature space is used for both the training and testing data, which can lead to overfitting or poor generalization of the machine learning model on new, unseen data.

To avoid this problem, I split the data into separate training and testing sets using the **train_test_split()** function from **scikit-learn**. This ensures that the training and testing data are independent and have different feature spaces. The **CountVectorizer** object is fitted to the training data only, and the resulting object is used to transform both the training and testing data separately. This way, the model learns to generalize well to new, unseen data, which is the ultimate goal of machine learning.

```
#extract the sentiment values
y_train = df_train['sentiment']
y_test = df_test['sentiment'].values
```

These two lines of code extract the sentiment values from the **sentiment** column of the **df_train** and **df_test** dataframes and store them in the **y_train** and **y_test** variables. **y_train** is a pandas Series object, while **y_test** is a numpy array.

```
#split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

This line of code uses the **train_test_split()** function from scikit-learn to split the data into training and testing sets. The function takes four arguments: the data to be split (**X_train** and **y_train**), the size of the testing set (20% of the total data in this case), and a random seed (**random_state=42**) that ensures the split is reproducible. The function returns four variables: **X_train**, **X_test**, **y_train**, and **y_test**, which are the training and testing sets of the text data and sentiment values, respectively. These variables will be used to train and evaluate machine learning models.

V. Classification model

a. Creation of classification models

The next step is to train a machine learning model to predict the sentiment labels for new, unseen data. With scikit-learn, the process will always be the same:

- create a model by indicating the desired parameters: here a classifier
- do the training thanks to 'fit', by providing it with the training data X and y
- predict results on the desired dataset with 'predict' from X data
- call the different desired metrics with the expected results and the predicted data as parameters

During this part, a lot of problems occurred:

For example, for the Naives Bayes and random Forest models, I had a problem with the *type* of data to pass in the function : the data passed as input to a function requires dense data, but sparse data is provided because I was working with sparse data with using text data with **CountVectorizer**. To fix this, I needed to convert the sparse matrix to a dense matrix. For example, for the Random Forest model, I convert the sparse matrix **X_train** to a dense matrix using the **toarray()** method of the sparse matrix before passing it to the **fit** method of the **RandomForestClassifier**. I also convert the sparse matrix **X_test** to a dense matrix using the **toarray()** method before passing it to the **predict** method of the **RandomForestClassifier**.

One other error with the SVM model was that the support vector machine (SVM) model failed to converge during training. SVM is a type of model that tries to find the best hyperplane that separates data into different classes. To fix the issue, I have increase the maximum number of iterations by setting the **max_iter** parameter in the SVC class to a larger value.

A big problem persists throughout the creation of the models: the compilation time. Apart from the logistic regression model, the compilation takes a lot of time, and some of them just run and I can't see the result even if there is no coding error... This stopped me to work on some models, because I couldn't figure out what was wrong, or just get the results. So for the knn model and naives bayes, I can't get the results. I left the model codes as comments in the script.

b. Training models: measurement and comparison of the results

Now let's see the results of the models carried out. I check the accuracy, recall and precision of each model to keep only the two most reliable.

By gradually deepening the function to clean the data, I managed to increase the accuracy by about 3-4%

In the theoretical part, it was said that the stronger the recall, the fewer errors there are.

```
LOGISTIC REGRESSION MODEL

Model logistic regression accuracy: 74.2546875
Confusion matrix :
 [[110285  49209]
 [ 33176 127330]]
Classification report :
              precision    recall  f1-score   support

           0       0.77      0.69      0.73    159494
           4       0.72      0.79      0.76    160506

    accuracy                           0.74    320000
   macro avg       0.75      0.74      0.74    320000
weighted avg       0.74      0.74      0.74    320000
```

For the logistic regression model: we can read that out of 320,000 tweets, 159494 (110285+49209, that can be seen in the support column of the classification report) are really negative and 160506 (33176+127330) are positive. However, the tested model only predicted 127330 tweets out of 160506 as truly positive. For the other 33176 he predicted negative sentiment.

We can see the results for the Random Forest Model and SVC model:

```
RAND FOREST MODEL :

Random Forest model accuracy: 68.68843749999999
Confusion Matrix :
 [[ 87487  72007]
 [ 28190 132316]]
Classification report :
              precision    recall  f1-score   support

           0       0.76      0.55      0.64    159494
           4       0.65      0.82      0.73    160506

    accuracy                           0.69    320000
   macro avg       0.70      0.69      0.68    320000
weighted avg       0.70      0.69      0.68    320000
```

For the Random Forest model: we can read that out of 320,000 tweets, 159494 (110285+49209, that can be seen in the support column of the classification report) are really negative and 160506 (33176+127330) are positive. However, the tested model only predicted 132316 tweets out of 160506 as truly positive. For the other 28190 he predicted negative sentiment.

```
SVC MODEL :

SVC accuracy: 74.1928125
[[109185  50309]
 [ 32274 128232]]
              precision    recall  f1-score   support

           0       0.77      0.68      0.73    159494
           4       0.72      0.80      0.76    160506

    accuracy                           0.74    320000
   macro avg       0.75      0.74      0.74    320000
weighted avg       0.74      0.74      0.74    320000
```

For the SVC model, the tested model predicted 128232 tweets out of 160506 as truly positive. For the other 32274 he predicted negative sentiment.

As we can compare, accuracy(Logistic regression)>accurace(SVC)>accuracy(Random Forest)

We can also see: (f1-score(Logistic regression)=f1_score(SVC)) >f1-score(Random Forest)

c. List of the two most trustworthy final classifier

The two classifiers that I retain are the logistic regression and the support vector classifier. These are the ones that have the best results, which are therefore more reliable and which will allow us to predict the sentiments of new tweets with the least possible error.

VI. Conclusion

During this project, I have completed a sentiment analysis project by following various steps such as data collection, data preprocessing, data preparation, visualization, model building, and model evaluation. I learned how to use different machine learning algorithms like Logistic Regression, Support Vector Machines (SVM), and Random Forest to train some model and predict the sentiment of tweets even if there is still some problem, but I have also encountered some problems during the process, such as converting the sparse matrix to a dense matrix and the SVM model failing to converge during training. However, I have successfully resolved these issues and evaluated the performance of the model using metrics such as accuracy, precision, recall, and F1-score. Overall, the project allows me a comprehensive understanding of the various aspects of a typical data science project.

VIII. References

Websites:

- Pandas. (2023). via NumFOCUS, Inc. Hosted by OVHcloud. Retrieved from https://pandas.pydata.org/docs/
- Stack Overflow. (2023). Retrieved from https://stackoverflow.com/
- DataCamp. (2023). Retrieved from https://www.datacamp.com/

Books:

- Géron (2019). “Machine Learning avec Scikit-Learn”. Oreilly
- Mathivet (2020). “Machine Learning : Implementation en Python avec Scikit-learn ”. Edition ENI