Achtatou Sean
Siwen Guo

University of Luxembourg

Bachelor in Computer Science

The Machine Learning
for sentiment analysis

Semester 3 - 2018-2019

1.Introduction

This Project has been done by Achtatou Sean, a student from the formation “Bachelor in Computer Science [BICS]” headed by the Prof.Dr.Nicolas Guelfi, in the University of Luxembourg for a biannual Bachelor Semester Project. The student has been helped by his Project Associate Teacher[PAT] Siwen Guo to manage to achieve the goals provided.

This Project is focusing on one of the many large domains existing and being studied in the Computer Science by many scientists. This domain, which is growing day after days, and being used daily in the Worldwide to solve many problems and helps the population in their daily tasks, is obviously the Artificial Intelligence. However, the Artificial Intelligence is still a vast domain to work with, since it is containing the following sub-domains: Machine Learning and the Deep Learning, which are used differently according to the type of problems we are dealing with. For this Project we are mainly focusing on the Machine Learning, which is preferred for the beginner in the Artificial Intelligence researches since it is more based on the statistics and prediction, so we have a better control of the data behaving

Beside, the Machine Learning is composed of many different techniques, used according to what type of data we wish to work with, to provide an efficient Artificial Intelligence algorithm. Here we are focusing on the Support Vector Machine [SVM] technique which is a Machine Learning algorithm being used on data to classify them, such as the cake is blue or red, or the messages have a positive or negative sentiment.

Indeed, for this project we will use the Support Vector Machine as a sentiment analysis, meaning it will be able to analyse the sentiment (positive, negative) of a set of messages, to create the base model. This model will have the possibility in the future, to determine by itself, if a certain new given message is acting as a positive or negative behaviour.

The Machine Learning algorithm will be implemented in a well known programming language: Python. Indeed, Python is a great programming language for the beginner programmers since it is simple to use and well flexible. Moreover, Python is progressively being used to create Artificial Intelligence in the Worldwide.

2.Project Description

2.1. Domain

The domain of the project is about the use of intelligent systems to classify information. The intelligent systems are being more and more used in the worldwide to accompany the humans in their daily life. Indeed, nowadays, there exist a lot of possibilities of tasks which can be handled by intelligent systems; going from drive a car to analyse people faces. However, one interesting field of the use of intelligent systems is the classification of data.

The classification can turn out to be very helpful in many situation; it can decrease the time taken to classify any type of information which would normally take hours for humans; it can also allow an intelligent system to perform tasks instead of a human, offering consequently the human additional time to achieve other tasks.

2.2. Objectives

The main objective of this project, is to understand how and why the concept of Artificial Intelligence is working, as it is being used in several companies, industries or specifics applications such as Google or Apple with Siri. To achieve this, we will have to focus on the implementation of an Artificial Intelligence. For this, we will have the possibility to implement our own program using the Machine Learning, which will be based on the existing technique of the Support Vector Machine.

This Machine Learning will have the capability to analyse messages, and return the ones with possible terrorism behaviour. Thanks to this, we will be able to analyze the behaviour of each part of the program, from the bottom to the top, and provide detailed explanation about each of the functionalities provided to manage an efficient intelligent system.

The second objective of this project, is to provide to the mankind a trustworthy regarding the use of Artificial Intelligence systems in their future daily life. Why the people should trust such intelligent programs being able to achieve tasks by themselves without any human action? To be able to convince the people, we will be able to show, thanks to the implemented program done in our main objective, that the programs using Artificial Intelligence can be trusted for some situations, however for others cases, it would be preferable to avoid using one.

And finally the last, but not the least, objective of this project, is to wide spread the knowledge about the Artificial Intelligence domain to the mankind. So the population stay up to date of the advance of this technology.

2.3. Constraints

As usual, a project can encounter some constraints, which can lead to it is whole misunderstand if not enumerated. This project is not an exception. The following constraints need to be acknowledged before moving further in the project.

The first constraint is obviously the basics mathematical background needed. Indeed, the Artificial Intelligence algorithm is mainly based on a multiples of mathematical equations, involving the domains of statistics, discrete mathematics, probabilities and prediction. If one of those domains is totally unknown, the following of the project might become rapidly difficult to fully understand. Otherwise, the project should seem to be easily understandable.

The second constraint is the programming background. Indeed, as said in the main objective of the project description, we will have to implement a program using one of the Artificial Intelligence branch, being the Machine Learning. To achieve this, we will need a programming language, for this project we have chosen Python for its simplicity. So, we need to know how to work and implement with the programming language Python, or at least having worked with another programming language and being able to switch to the Python language.

The last constraint, but not the least, if possible, is the possibility of already having some background in the Artificial Intelligence domain. Indeed, the Artificial Intelligence is a concept which is hard to understand at first sight because of its complexity and new way of thinking with data. For someone with no background, it could take much longer to understand, but would still be able after while to follow.

3.Background

3.1. Scientific

For the scientific part, we will talk about the real operation of an Artificial Intelligence. We will carry on with the different types of Artificial intelligence we can work with such as Supervised, Unsupervised or semi-supervised. Finally, we will conclude with the description of the two sub-domains of the Artificial Intelligence namely the Deep Learning and Machine Learning, and why we will use the Machine Learning.

3.1.1. The Artificial Intelligence

First of all, what is the Artificial Intelligence? Most of the population, think that the Artificial Intelligence, being used in computer science, are machines with the mentality of acting intelligently like a human. But, it is certainly not the case. Indeed, the Artificial Intelligence is mainly based on several mathematics concepts in the domains of prediction, statistics and probabilities. The Artificial Intelligence, is a program aiming to predict multiples events in the future based on its actual knowledge and analyzing, to achieve the goals it has been designed to solve. There exist a multitudes of domains in which the Artificial Intelligence is used according to the need of the human, such as:

- Analyzing of words/images
- Manipulation of objects
- Driving a car
- Having a conversation with a human
- …

3.1.2. Types of learning

Afterwards, the Artificial Intelligence can be used on three different types of learning:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning

The Supervised learning, is a learning using a function mapping from an input to an output, based on multiples input-output examples. The input-output examples are called training set, they contain the input like a sentence, and the already known output like the sentiment of the sentence. Those are called, data already classified or labelled - supervised. Each of the single input-output example will be represented as a vector. Later, a Supervised learning algorithm will take all the training set - vectors - and produce a fixed function in a model in which an input-output example will be represented as a dot. This model will be used to map new input - test set - and predict their output according to the actual model. This type of learning is mainly used in Artificial Intelligence for classification and regression.

The Unsupervised learning, is the inverse of the Supervised learning, namely, instead of learning on already labeled or classified data such as the Supervised learning does, the Unsupervised learning will learn to identify the commonalities in the input data to create the model - unsupervised. Whenever it will receive new piece of data, it will behave differently based on the absence or presence of these commonalities of these new data. The Unsupervised learning is well known to be used for the Neural networks.

The semi-supervised falls in between the two. This is the combination of both Supervised and Unsupervised data, meaning it will use some labelled data, and unlabelled data to create a model.

3.1.3. Machine Learning versus Deep Learning

Finally, we will focus on the difference between the Machine Learning and the Deep Learning in the Artificial Intelligence.

The Machine Learning, being a subset of the Artificial Intelligence, is based on the building of models based on sample data (trained data) to be able to make prediction on new data using a computer, without the help of a human, so it is a Supervised learning. It is used in multiples applications such as:

- Email filtering
- Classification
- Regression

The Machine learning particularity, is its ability to manage to work efficiently with only thousands of data points (vectors) as input. But, on the other side, the only type of result we can obtain, namely the outputs, will be numerical value.

For the Deep Learning, being a subset of the Machine Learning, it is based on the learning of data representation. It means, it is automatically capable to discover itself the needed representations for the

detection of features, or the classification of data. Namely, it does not need sample data such as the Machine Learning does. The Deep Learning operation is mainly based on the use of neural networks architecture and can be used for:

- Speech/Audio recognition
- Image analysis

However, compared to the Machine Learning, the Deep Learning needs to use big data, meaning millions of data points to be optimal. But the type of result it can obtain are much larger, such as :

- Sound
- Word
- Image

For this project, we are working on the Sentiment Analysis of sentences to detect if they are containing possible terrorism treats or not. Therefore, the use of the Machine Learning seems reasonable. Still, the Machine Learning can be used with different technique, we decided to use the Support Vector Machine which will be detailed in the technical part.

3.2.Technical

For the technical part, we will talk about the technique used in the Machine Learning for the project, namely the Support Vector Machine. We will proceed with the explanation of how and why the preparation of the data is mandatory before considering the use of the Support Vector Machine algorithm.

3.2.1. The Support Vector Machine

The Support Vector Machine, is a Supervised Learning model used for the classification of data. By having a set of training examples already classified, the Support Vector Machine will take the set of training examples - vectors - and fit it in a model; which will be used as classifier in the future. In the model, each examples of the training set, is represented as a set of dots in the space with its proper category.

If linearly separable, the Support vector machine will divide each type of category by a clear gap - line - which can be represented by hyperplanes**[Picture 3]**. This gap has to be as wide as possible from the two category, and can be calculated by the margins around the hyperplanes.

The Support Vector Machine as explained above, is using linear classification to divide, but it can work with non-linear classification as well, using a kernel trick. Instead of working on two dimensional, the Support Vector Machine will work on three dimensional feature spaces. And in this case, it will be capable to divide the category with hyperplanes of three dimension.

3.2.2.Introduction to pre-processing

Now, we will focus on the preparation of the data. As said above, the Support Vector Machine need some training data represented as vectors to fit in the model, to train it; but it can not take any kind of data. Indeed, it can not take a clear messages of type string to create a vectors, it needs integer values. This is where the preparation of the dataset is needed.

The preparation of the data set requires multiples steps to be able to create effective vectors to fit in the model.

The first step is to get the dataset. Indeed, the whole dataset which we would like to work with, need to be fully imported.

The second step, is to handle all the possible errors or missing data. It is possible that some data might have been corrupted or are simply missing. In this case, we need to take care of that, because the model might miss predict those values. We can have multiples possibilities of corrections:

- Add the value “0”, where the data is missing or corrupted.
- Rewrite the missing or corrupted values.
- Don’t take the value, and continue with the next values.

The last step will be the pre-processing of the data; it will allow data to be represented under the form of vectors which will be fitted in the model. The pre-processing regroups multiples step:

- Split the sentence into sentences tokens
- Get rid of special characters in the sentences tokens
- Split the sentences tokens into words tokens

- Discard the stop words into the words tokens
- Lemmatize the words

The tokens of each sentences will represent all the features which will be used to create the vectors.

3.2.3. The Glove library

Another possibility to create a vector, is to use the extension Glove. This extension already provides vectors of fixed small size for most of the words. A vector for a sentence, would simply be the sum of his features.

However, we will talk more about the pre-processing in details during the implementation of the Machine Learning.

4. Machine Learning for Sentiment analysis

4.1.Requirements

For the requirements we will talk about all the tools and materials we need to consider to be able to realise, the most efficiently, the project being the implementation of a Machine Learning for sentiment analysis in the homeland security. As summary, we will be concerned about the programming language used, the importation of libraries, files, and how the Machine Learning implementation will be designed. **[Picture 1]**

4.1.1 The programming language Python

First to first, we will focus on the type of programming language that has to be used for this project. The Python 3 language seemed to be a good choice. Indeed, Python is a considered as a pretty good language, and is getting known in the worldwide. Its main particularities are being its flexibility and simplicity to be used by any programmer, belonging to the amateur or professional programming, it is very simple to learn how it is working.

Moreover, this language is beginning to be used for Artificial Intelligence since few years from now, ever since Python is said to be a very powerful language.

4.1.2 The libraries

In the second part, this project requires multiples libraries to accomplish (nltk, numpy, pandas, scikit-learn, glove, scipy, seaborn, re, csv)**[Picture 2-10-31]**. Each library has a specified role to do, and will be used all along the project. Among those libraries, we can notice:

- Nltk (natural language toolkit):
  - ► This library is used to facilitate the work with human language data. It allows the tokenization of the data and provides an interface easy to use, for doing the pre-processing of the data.
- Numpy:
  - ►This library is used to allow easy computing in Python. Numpy has the capability to play with n-dimensional array and establish transformations into these. But it can also be used as a simple container of data.
- Pandas:
  - ►This library is used to ease the use of data structures and data analysis; it allows the modelling

■ Scikit-learn:

►This library regroups all the possible techniques which can be used to create a Machine Learning, to work with data.

■ Glove:

►This library, based on an unsupervised algorithm, is used to obtain vector representation of words.

4.1.3. The data

In the third part, we will talk about the type of data desired and to import for a project. Indeed, to be able to realise a Machine Learning, we need some kind of data with is correlated to what we wish to classify. Obviously the type of data, depends on what type of analysis we would like to do.

As example, if we wish to classify cats and dogs, we will use data with details such as the weight, the height, the food eaten and so on; and already classified with the type of animal it refers to. However, if we wish to classify sentences based on the sentiment of the person writing being bad or good, we will need data with multiples different sentences and the sentiment classified next to it.

Moreover, with Supervised Machine Learning, since the data is already classified, a person must have already done it manually and provided the classification of the data. But, the issue is that it can lead to some errors in the classification, if the researcher think a sentence is good while it is bad, nay very bad if the classification has been done by an Artificial Intelligence. Most of the time, the data is imported as an ".csv" file, which allows a better maneuverability of the data, since they can be spaced by a comma, and more readable.

4.1.4. The pre-processing

For the fourth part, we will review in details the pre-processing step which has to be done on the data. Once, we have imported the data we want, we need to pre-process all of it. The pre-processing is a mandatory step, without this step it would be impossible to create the vectors of the data which has to be fitted in the model. The pre-processing is using two important libraries, in particular "nltk" and "re". Moreover, the pre-processing can be done with or without the use of the library "Glove", the two methods will be discussed below.

The first method would be without the use of "Glove", meaning the vectors are manually done. Let us pre-process as example a sentence – "@julierest I like playing football in the backyard. This is funny #FUN ". The following will show the pre-processing steps on the sentence:

■ 1. The message will be split in tokens of sentences with use of the library "nltk".

►The full sentence "@julierest I like playing football in the backyard. This is funny #FUN", will become two tokens being: "@julierest I like playing football in the backyard." and "This is funny #FUN".

■ 2. The tokens will get rid of the regular expressions such as "@, (, /, #", using the library "re".

►The tokens will become: "I like playing football in the backyard." and "This is funny", since "@julierest" and "#FUN" are considered as noisy for the data.

■ 3. The tokens will be split as words with the use of the library "nltk.

►We will have the tokens:
"I","like","playing","football"," in" ,"the" ,"backyard" ,"This", "is","funny".

■ 4. Each token being a stop words will be discard. The stop words are words being common in a certain language because the have a high probability to be used in any type of sentence, therefore they could lead to noisy data and must be discard.

►We will have the tokens:
"like","playing","football"," in" ,"backyard","funny".

■ 5. The next step is the lemmatization, it will simply slightly transform the token such that they are represented at the first-person singular and low-cased. The tokens are hopefully already in this form.

The tokens that we will get for each sentence at the end, will be the features. All the features of each sentence together will be used to create each vector for each sentence. Each time a feature appears in the sentence, we will increase the axe by "1". Unfortunately, the numbers of features will represent the length of the vectors, which can lead to vectors of big length and extend the time taken to fit them in the model. If we assume a sentence is giving five features at the end, and we need to pre-process five-thousands of sentences, we can end up with vectors of length around a thousand. **[Picture 9]**

The second method would be to use “Glove” library. The advantage to use this library is the ability to fix the length of the vector whatever is the sentence, thus it would take less time to fit in the model. The steps used are exactly the same than the ones enumerated above just before. However, the vectors of the sentences are not created based on all the features of all the sentences, but only on those specific to the sentence. More clearly, the “Glove” library provide thousands of words having a fix vector. Whenever we want to create a vector for a sentence, we take the features of the sentence and add the vector of each of those words giving the final vector for the sentence. Moreover, the “Glove” library can provide different length of vector such as fifty, hundred or hundred and fifty according to the need.
[Picture 8]

4.1.5. Linear model versus kernel model

The last part of the requirements we need to focus on is the Support Vector Machine type of models. Indeed, the Support Vector Machine can work linearly or using the kernel trick to create a model. However, by using the linear type, it could happen that the vectors fitted in a model can simply not be linearly separated, giving bad classification. In this case, the Support Vector Machine can provide the use of the kernel trick. The kernel trick will allow the possibility to work in a three dimensional model, which will solve the problem we had with linear, since it was evolving in only a two dimensional model.

4.2.Design

For the design part, we will explain in details how we did use the requirements, to do the implementation of a Machine Learning using the Support Vector Machine. The design part will be divided in two different sections; the first section will be regarding the implementation of a Machine Learning without the use of the external library “Glove”, while the second section will provide the use of the “Glove” library. Moreover, for both sections, we will each time provide two different models, a linear one and kernel trick one.

Recall that for this project, the objective is to provide Machine Learning models for “Sentiment Analysis”. For this we will work with two imported filed containing thousands of different messages already classified, meaning we know the sentiment of each messages. The first file will focus on message coming from specifics companies such as Apple, Google, Microsoft and Twitter. The second file will contain global messages and we will have to provide a domain adaptation, we will have to choose ourselves the more interesting messages according to the domain we work with, meaning for us the detection of terrorism activities.

4.2.1. Machine Learning models on the corpus “corpus-medium.txt”

Let us begin with the first section, providing two models without “Glove” and focusing on specific messages.
[Picture 12]

We have to first import all the needed libraries (nltk, numpy, pandas, scikit-learn, glove, scipy, seaborn, re, csv). A solution would be to do it manually with the command “-pip install <library>” from the console of Python. However, if not comfortable with this method, we can do it automatically as well, with the following selection in the Python application [File→Settings→Project→ProjectInterpreter→+→Enter the library to download in the search bar→Select it→Install Packages]. After having installed all the necessary packages, we can use them all along the project.

Afterwards, import the files of data containing the messages and read them. The file used is named “corpus-text-medium.text”, it is a “.csv” type file and contains a thousand messages from Apple, Microsoft, Twitter and Google already classified. To read the file, we need to use a delimiter “,”, to provide to the computer a method to detect each part of the file. For each line of the file, we have to get the second and fourth rows containing the message and the sentiment, while the other rows are not used. We have to read all the file, and put the message in an array1, and the sentiment to these messages in an array2. **[Picture 21]** Nevertheless, the standard array size of Python is limited, since we are working with big data, we might need to maximize the size of these array.
[Picture 11]

Once read and acquired all the messages, each messages in the array1 will be pre-processed to provide the features used to create the vectors. **[Picture 18]** For the pre-processing, we have to pre-process individually each messages and add their features in an array3. The pre-processing will execute the following on each message:

► If the message contains multiple sentences, each of these will be split as multiples tokens sentences.
► Each sentences will be checked. The check will get rid of the useless and noisy data such as "@...,(…),#,.." and all specials characters.
►Each of the token will be split again in words tokens.
► The words are checked to be stop words, if there are, the are discard.
► The last step is the lemmatization, which will slightly transform the tokens word in lowercase and in singular.
► The tokens are returned.

For each messages, the tokens returned, are called features, and will be added to the array3, which will frequently give us a big array. **[Picture 26]**

Once all the features are in the array3, we begin to create the vectors for each messages. **[Picture 33]** Each messages will be given a vector of the length corresponding to the number of features in the array3 with all zeros. Once done, we have to search through all the array3 to check if a certain word is appearing in the message. If it is true, the position of the word in the array3, will be correlated with the position of the vector of the message, and will be added one, otherwise it will stay zero. Once, a message has been checked through all the array3, we can save the vector in a file ("Data-to-be-filled-no_glove.txt"), containing as first row, the head with the sentiment and all the tokens of the array3. For each line we will save the vector of the message, with its sentiment which can be found in the array2. In the end, the file will contain all the vectors for each messages. **[Picture 13-20]**

Nevertheless, we have to provide to the machine a way to understand and use our vectors. To achieve that, we can represent the vectors as matrices. We will have two matrices, one used for the vectors of the messages, and the other one used for the sentiment of each vectors. **[Picture 15-16-24]** However, the sentiments are of type "positive" and "negative" being strings, which is not understandable and usable for the model since it only takes integer. To deal with this problem, we have to change the strings to integer, in this case we let positive being zero, and negative being one. **[Picture 25]**

Now, we are able to use the matrices to create a training and test set. For this project, we used as training set eighty percent of the messages and twenty percent for the test set. Because, most of the time, more we have training examples, more our model will provide a better prediction; but the general condition is to have a training set containing more than half of the data. Nonetheless we still have to randomize the position of the messages since they are ordered, otherwise we would have as training set only positive messages. To accomplish this, we randomize the chosen vectors. The "X_train" represent random vectors in the first matrix, and "Y_train" the random sentiment in the second matrix of each messages in the "X_train", those will be fitted in the model. The "X_test" and "Y_test" represent the rest of the messages and their sentiment which will be used to test the model.

Now, we have to create our models. The first model is the linear one, taking as parameters:

- "kernel = "linear""
- "C = 10"
- "class_weight = "balanced""

The second model is the kernel one taking as parameters:

- "kernel = "rbf""
- "C = 2"
- "class_weight = "balanced""
- "gamma = 0.01"

Once, the models are created, we can fit both of them with "X_train" and "Y_train". It will input in the model each of the vectors of the messages with their sentiment, and create the hyperplane. When fitted, ours models are now ready to predict new income data. We may use the "X_test" and "Y_test" to test the accuracy of each models, so we can get an idea of the capability of prediction for each of them. Now, that our models are ready, we can give to each one only the "X_test" representing vectors, and the model will output the prediction of the sentiment of each of these vectors in "X_test" according to where they will end up laying in the model. **[Picture 14]**

4.2.2. Machine Learning with the corpus "corpus-text-1m600.txt"

Let us now focus on the second section, which will provide two other models, but operate on "Glove", and perform with global messages which has at first to be sorted.

As the first section, we need to import the file containing the data we are going to work with. The file used is named “corpus-text-1m600.txt” and is as well a “.csv” file. The file contains one million and six hundred messages already classified, however the classification is slightly different, a zero will be positive and a four will be negative. **[Picture 27]**

Like the first section, we have to put in arrays the messages and their sentiment. However, we need to first sort the messages which are considered as useful for the domain we are working with. Since the domain is the detection of terrorism activities, we will have to select the messages containing words related to terrorism, words which will have to be chosen by ourselves. The messages got at the end, will be put in an array5 and their sentiment in an array6. **[Picture 28-38]**

Thereafter, the pre-processing of each messages in array5 is exactly the same than the one used for the first section. However, during the creation of the vectors, we will use “Glove”. **[Picture 36-39]**Instead of using all the features of all the messages, a message will only use its own features to create its vector. For this project we decided to use “Glove” manually, meaning we imported its library containing all the words, so we could understand and manipulate precisely the vectors. Thus, for each features got from the pre-process of a message, we have to go through the library of Glove to get the feature vector, the vectors got will be added together giving the final vector of the message. This process has to be done for each messages, and the vectors got will be saved as in the first section in a file (“Data-to-be-filled-possible-terrorism-with_glove”) containing for each line the sentiment of the message and its vector. **[Picture 29-37]**

The representation as matrices of the array are almost the same as for the first section, except we do not need to modify the sentiment since it is already as integer.

The method of creation of “X_train”,”X_test”,”Y_train”,”Y_test”, are likewise the same than those explained in the first section, except we are using seventy percent of the data for the training set and thirty percent of the data for the testing set, because we have much more data.

Finally, the fitting of the model is evenly almost the same as the first section, except the parameter C for the linear model is now five, and the parameter C for the kernel model is three.
[Picture 30]

Remark, after having created each of our four models, we have the possibility to save each of them, so we gain the capability to use the models to perform a prediction on future new data. To be able to do this, we can use the external library “sklearn” by importing “joblib”. This library receives as inputs; the model created, and the name given to it, and will save the model in the project. **[Picture 23-35]**

4.3.Production

For the production part of this project on the Machine Learning, we will be concerned about all the different results we acquired from the design part for each of the models. Namely, we will concentrate on the observation of the final models we created earlier. We will view each model accuracy and their potential of predictability we can obtain. To accomplish this, we will focus on each of the four models we created in the design part for the project and compare them.

4.3.1. Get the prediction and the accuracy of the models

Foremost, we get to observe the result we received for the Machine Learning models, using the “corpus-text-medium.txt” data, and without using the glove library. But first, we need to explain how to get the accuracy and the prediction of the models **[Picture 41]**.

To get the accuracy and the potential of predictability of the models, it is necessary to use an external library. The accuracy of the model, which is the possibility for the model to provide a good prediction, can be determined by taking the score of the model, which is accepting as inputs the “X_test” and “Y_test”, being the test sets. The following score will output an accuracy based on the inputs. Since the model knows the sentiment of the “X_test” set due to the “Y_test” set, whenever some “X_test” data obtain a wrong “Y_test” sentiment, the model will acknowledge it, and the accuracy will drop. In most of the case, we could assume a model to be a good if its accuracy is included between seventy-five and ninety-five percent, otherwise the model may result in a bad prediction.

4.3.2. Result for the models on the corpus “corpus-text-medium.txt”

Still, for the linear model we obtained an accuracy of eighty-two percent.**[Picture 19]** Since we now know the accuracy is rather large, we can aim to use the model to predict on the "X_test" data. The prediction will output the sentiment of the data based on the accuracy of the model, since we know the accuracy is not perfect, some outputs have to be wrong.

Moreover, to obtain an improved view of the power of prediction, we can compare the prediction obtained of the model with the real data we should get as output.**[Picture 17]** Through that possibility, we will be able to calculate with precision the probability to get a good result or not. To do this, we have to calculate the number of times a sentiment is correct and incorrect, and divide each by the numbers of data. By calculating, it gives us the following prediction table. **[Picture 4]**

We can observe with the table, that with this model, we have a high possibility to obtain a correct prediction, being showed by "True-Positive" and "True-Negative". While, the number of wrong prediction, "False-Positive" and "False-Negative", may be considered as rather low. **[Picture 22]**

Now, we can focus on the next model, which is still using the "corpus-text-medium.txt", without the use of the Glove library, but is this time using the kernel trick. Likewise, to get the accuracy and prediction of the model is the same technique used than for the last model. However, for this model, we obtained an accuracy of eighty-six percent, which is slightly higher than for the last model not using Glove. Moreover, by observing the table of prediction for this model, we get this table. **[Picture 5]**

.

We can observe with the table, that with this model using Glove, we have a better and higher possibility of correct prediction, being showed by "True-Positive" and "True-Negative", than for the last model. Moreover, the number of wrong prediction, "False-Positive" and "False-Negative", are also lower than for the last model.

4.3.2. Result for the models on the corpus "corpus-text-1m600.txt"

In the second part, we are getting interested in the result for the Machine Learning models using the "corpus-text-1m600.txt", with which we realised a domain adaptation by focusing on the messages having a possible terrorism behaviour. Likewise, the two previous models, we can calculate the accuracy and the prediction use the same manner but with the "X_test" and "Y_test" of the "corpus-text-1m600.txt" data.

First of all, we begun with the linear model not using the Glove library. During, the calculation of the accuracy we only obtained sixty-one percent. **[Picture 32]**Matter, this result can be explained by the fact that we needed to choose ourselves the messages by given words belonging to terrorism behaviour, which can lead to some noises in the data if the words are badly chosen or overly global. Thus the accuracy of the models could eventually drop. Nevertheless, by looking at the prediction table. **[Picture 6]**

We can observe, aside the accuracy, that the possibility to obtain a correct prediction for positive behaviour is slightly lower than for the first model not using Glove, but still correct. However, a correct prediction for a negative behaviour has widely dropped compared to the last model not using Glove, which is unacceptable since we need to predict a possible bad behaviour in the data.

Afterwards, for the model using the kernel trick and using the Glove library, we got the result for the accuracy of sixty-four percent, which is slightly higher than the last model. Moreover, by looking at the prediction table. **[Picture 7]**

We can observe with this model, aside the accuracy, that the possibility to obtain a correct prediction is almost the same than for the first model not using Glove. Although, the interesting part is the fact this model has obtained the better probability to predict a bad behaviour compared to all four models. Which is great since we need the model to predict all the possible bad behaviour of the data. **[Picture 34]**
[Picture 40]

4.4.Assesment

For the assessment part of this project, we will have the possibility to review about the different result we obtained in the production part regarding all the four different models. We will put forward the power of predictability of our four models and discuss about their good and bad aspects.

For the first model, which was using the first corpus “corpus-medium.text”, without the use of the Glove library. We can clearly observe that the calculation for the accuracy of the model, which is of eighty-two percent, can be considered as satisfactory. Indeed, when we did calculate the prediction power and the prediction table of the model, we could see that its power of prediction was high. Those last results ensure that this model could be already capable to provide a good prediction to detect the sentiment of a message if used in a real world situation.

For the second model, which was still using the first corpus without the Glove library, but using the kernel trick. We can observe that, by using the kernel trick instead of a linear model, the prediction accuracy of the model slightly increased. This increase ensued a better result in the prediction table, and consequently increased the power of prediction of the model to be able to detect good or bad sentiment of messages. Those last results ensure that this model could be used in a real world to predict messages sentiment as well.

For the third model, which was this time using the second corpus “corpus-1m600.txt”, focusing on terrorism behaviour, and with the use of the Glove library. We obtained an accuracy of only sixty-one percent, which at first sight might presume that the model could have a bad prediction power. This result can be explained as we had to choose by ourselves the messages based on words which could have been related to terrorism behaviour. But it can as well bring noises to the data and altered the final result as it did. However, when we calculated the prediction table, the prediction was still high but slightly lower than for the first model. Nevertheless, we can observe that this model has a great prediction to predict good behaviour, however it is poor to predict the bad behaviour since the possibility to get a great prediction for a bad behaviour is only of seventy-two percent.

For the last model, still using the second corpus focusing on terrorism behaviour and using the Glove, but this time using the kernel trick. We observed an accuracy of prediction of sixty-four percent. This result can be explained by the same reason enumerated above for the last model. However, when we calculated the prediction table, we obtained an accuracy for a correct prediction of good and bad behaviour being above the ninety percent. Thus, those results could ensure that this model is great to predict a possible terrorism behaviour since the possibility of errors is well low.

Mainly, by working around the results of the four models, we could observe that by using the Glove library, the accuracy of the models was increasing and bringing a better prediction power for the model. This is explainable by the fact that the vectors of the model with Glove have a smaller length for the vectors than the ones not using Glove, providing a better accuracy.

However, we had only the possibility to compare the models two by two between them using different corpus of messages. We could have compared the four models between them using the same corpus, to observe the real difference between the use of different parameters on the models. Moreover, the models only predict the messages as they are, we are not considering the case the people could do irony, which is a situation that our models can not detect yet.

5.Conclusion

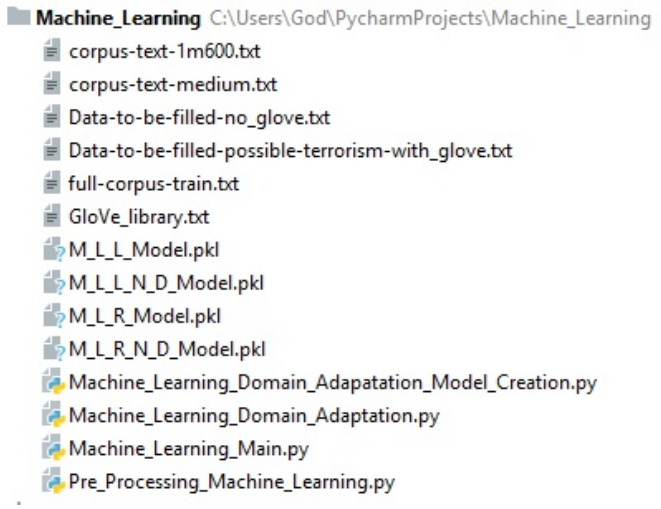
To conclude to this project. We have had the possibility to work with an Artificial Intelligence to provide a supervised Machine Learning, based on the Support Vector Machine technique, capable to detect the sentiment of messages based on two different corpuses. The first corpus was based on the sentiment of the messages and the another one based on the possible terrorism behaviour of the messages.

We could note that the Machine Learning could prove to be very efficient in several ways, and operate very nicely to predict the behaviour of messages by using different models, and thus could be used in real world situation. However, the Machine Learning could also result in a bad prediction for some others models and eventually not being used, as the result would deflect too much from the reality, which would lead to errors.

In any case, the Machine Learning models were resulting in a satisfactory accuracy and consequently providing a valuable prediction power overall. However, the models prediction could have still been ameliorated, indeed the actual models are not able to detect a possible irony in the analyzed messages, which could lead to a bad prediction from the models.

Nevertheless, a model will never provide a hundred percent accuracy, since errors could always occur in any situation, and the machine operation can some of the time be unpredictable.

6.Appendix



Picture 1. Representation of the final project application (corpus, files for vectors, Glove library, four models of Machine Learning saved, Machine Learning application, pre-processing application, domain-adaptation application)

```
import csv
import nltk
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns;sns.set(font_scale=1.2)

from time import sleep
from Pre_Processing_Machine_Learning import pre_processing
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib
```

Picture 2. Representation of the full imported libraries for the project to use the Machine Learning and Support Vector Machine technique.

To calculate the hyperplanes, we can use the following equations:

- **Legend**: $\overrightarrow{x_i}$ is the x value of each training data set $(\overrightarrow{x_1}, \overrightarrow{y_1}),\dots,(\overrightarrow{x_n}, \overrightarrow{y_n})$.
  $\vec{w}$ is the normal vector to the hyperplane.
  $\frac{b}{\|\vec{w}\|}$ is the offset of the hyperplane from the origin to $\vec{w}$, distance between hyperplanes.
  $y_i = 1,0,-1$

- $(\vec{w}.\overrightarrow{x_i}) - b = 1$ (Represent the hyperplane at maximized distance above the main hyperplane)
- $(\vec{w}.\overrightarrow{x_i}) - b = 0$ (Main hyperplane being in the middle of the gap, equally distant of each category)
- $(\vec{w}.\overrightarrow{x_i}) - b = -1$ (Represent the hyperplane at maximized distance below the main hyperplane)

To create a gap as wide as possible, we need to maximize the distance between the hyperplanes, so we need to minimize $\|\vec{w}\|$,the distance between a point to the hyperplane, which are called the margins. The margins can be calculated by using the formula of a distance from the origin to a point. We can let, the origin being the hyperplane; and the point, with coordinates $x, y, \dots$,being a dot of the model, giving the formula:

- $\sqrt{x^2 + y^2 + \cdots}$

However, we need to avoid any points to lay in the margins space, so we need to add constraints:

- If $y_i = 1 \rightarrow (\vec{w}.\overrightarrow{x_i}) - b \geq 1$
- If $y_i = -1 \rightarrow (\vec{w}.\overrightarrow{x_i}) - b \leq -1$

Therefore, we can globally for all the points, to minimize the distance use the formula:

- $y_i - (\overrightarrow{w}.\overrightarrow{x_i} - b) \geq 1$, $\forall\, 1 \leq i \leq n$

Picture 3. Representation of the full mathematical method used to find the hyperplane in the models using the Support Vector Machine.

Prediction/Sentiment	Positive	Negative
True	93 %	89 %
False	7 %	11 %

Picture 4. Results for the table prediction of the model linear with the corpus "corpus-medium.txt" without using Glove.

Prediction/Sentiment	Positive	Negative
True	96 %	98 %
False	4 %	2 %

Picture 5. Results for the table prediction of the model using the kernel trick with the corpus "corpus-medium.txt" without using Glove.

Prediction/Sentiment	Positive	Negative
True	89 %	72 %
False	11 %	28 %

Picture 6. Results for the table prediction of the model linear with the corpus "corpus-1m600.txt" using Glove.

Prediction/Sentiment	Positive	Negative
True	93 %	96 %
False	7 %	4 %

Picture 7. Results for the table prediction of the model using the kernel trick with the corpus "corpus-1m600.txt" using Glove.

"like"	0.36808,0.20834,-0.22319,0.046283,0.20098,0.27515,-0.77127,-0.76804,-0.34861,0.5052,... [50]

+

"eat"	-1.5236,-0,26458,0.27174,0.82746,-0.098719,0.040697,-0.56075,1.1149,-1.0156,0.46597,... [50]

=

Vector of : "I like to eat"	-1.15552,-0.05624,0.04855,0.873743,0.102261,0.315847,-1.33202,...[50]

Picture 8. Example of use of the library Glove with the features of a message to get the vector of the message based on these features.

Features/Sentences	like	rabbit	playing	football	shoot	funny	...
"@julie.rest I like playing football in the backyard. This is funny #FUN "	1	0	1	1	0	1	...
"Yesterday, I was playing with a rabbit and it got shot by a hunter"	0	1	1	0	1	0	...
...	...	...	...	...	...	...	...

Picture 9. Example of use of the set of features obtained from all the messages to create the vectors for those messages.

```
import csv
import sys
import pandas as pd

from time import sleep
from Pre_Processing_Machine_Learning import pre_processing
```

Picture 10. Representation of the imported libraries in Python 3 to use the Machine Learning with the Support Vector Machine.

```
csv.field_size_limit(sys.maxsize)
```

Picture 11. Representation of the line of implementation allowing to maximize the size of a list.

```
"Topic","Sentiment","TweetId","TweetDate","TweetText"
"apple","positive","126415614616154112","Tue Oct 18 21:53:25 +0000 2011","Now all @Apple has to do is get swype on the iphone and it will be crack. Iphone that is"
"apple","positive","126402758403305474","Tue Oct 18 21:02:20 +0000 2011","Hilarious @youtube video - guy does a duet with @apple 's Siri. Pretty much sums up the lov
"apple","positive","126397179614068736","Tue Oct 18 20:40:10 +0000 2011","@RIM you made it too easy for me to switch to @Apple iPhone. See ya!"
"apple","positive","126379685453119488","Tue Oct 18 19:30:39 +0000 2011","The 16 strangest things Siri has said so far. I am SOOO glad that @Apple gave Siri a sense
"apple","positive","126377656416612353","Tue Oct 18 19:22:35 +0000 2011","Great up close & personal event @Apple tonight in Regent St store!"
"apple","positive","126373779483004928","Tue Oct 18 19:07:11 +0000 2011","From which companies do you experience the best customer service aside from @zappos and @ap
"apple","positive","126366353757179904","Tue Oct 18 18:37:41 +0000 2011","Just apply for a job at @Apple, hope they call me lol"
"apple","positive","126365858481188864","Tue Oct 18 18:35:43 +0000 2011","Lmao I think @apple is onto something magical! I am DYING!!! haha. Siri suggested where to
"apple","positive","126360935509135362","Tue Oct 18 18:16:09 +0000 2011","RT @PhillipRowntree: Just registered as an @apple developer... Here's hoping I can actually
"apple","positive","126360398885687296","Tue Oct 18 18:14:01 +0000 2011","Wow. Great deals on refurbed #iPad (first gen) models. RT: Apple offers great deals on refu
"apple","positive","126358340220616704","Tue Oct 18 18:05:50 +0000 2011","Just registered as an @apple developer... Here's hoping I can actually do it... Any help, g
"apple","positive","126357982685569024","Tue Oct 18 18:04:25 +0000 2011","你好 ! Currently learning Mandarin for my upcoming trip to Hong Kong. I gotta hand it to @A
"apple","positive","126349695676203009","Tue Oct 18 17:31:29 +0000 2011","Thank you @apple for Find My Mac - just located and wiped my stolen Air. #smallvictory #thi
```

Picture 12. Representation of the corpus "corpus-text-medium.txt" containing the messages and sentiment of multiples users from Apple, Google, Microsoft and Twitter.

```
Sentiment,apple,get,swype,iphone,crack,hilarious,youtube,video,guy,duet,siri,pretty,much,sum,love,affair,rim,made,easy,switch,see,ya,16,strangest,thing,said,far,so

positive,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

positive,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

positive,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

positive,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

positive,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

positive,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

Picture 13. Representation of the file "Data-to-be-filled-no_glove.txt" containing the vectors with the sentiment of each messages after the pre-processing and the creation of the vectors for the corpus "corpus-text-medium.txt".

```
dataset = pd.read_csv("Data-to-be-filled-no_glove.txt")                         #Represent all out dat
dataset.head()

features_words = dataset[Set].as_matrix()   #Put the features as a matrix#
type_label_initial = dataset["Sentiment"].as_matrix()
type_label = np.where(type_label_initial=="positive",0,1)

print("Division of a set of vectors for training and testing.")
X_train, X_test, Y_train, Y_test = train_test_split(features_words,type_label , test_size = 0.2 ,random_state= 0)
sleep(2)

print("Creation of the linear model and the rbf model.")
model = svm.SVC(kernel="linear",C = 10, class_weight= "balanced")                #model for kernel = 
model2 = svm.SVC(kernel="rbf", C = 2, class_weight = "balanced", gamma = 0.01)   #mode for kernel = r
sleep(2)

print("Input of the training vectors in each model.")
model.fit(X_train,Y_train)
model2.fit(X_train,Y_train)                                   #Fit the models with the training da
```

Picture 14. Implementation, for the creation of the matrices for the vectors and sentiment from the file "Data-to-be-filled-no_glove.txt", for the set of training and testing vectors chosen randomly, the creation of the two models (linear and using the kernel trick, fitting of the two models with the training sets.

	Sentiment	apple	get	swype	iphone	crack
0	positive	1	1	1	1	1
1	positive	1	0	0	0	0
2	positive	1	0	0	1	0
3	positive	1	0	0	0	0
4	positive	1	0	0	0	0
5	positive	1	0	0	0	0
6	positive	1	0	0	0	0
7	positive	1	0	0	0	0
8	positive	1	0	0	0	0
9	positive	1	0	0	0	0
10	positive	1	0	0	0	0
11	positive	1	0	0	1	0
12	positive	1	0	0	0	0
13	positive	1	0	0	0	1
14	positive	1	0	0	0	0
15	positive	1	0	0	0	0
16	positive	1	0	0	0	0
17	positive	1	0	0	0	0
18	positive	1	0	0	0	0
19	positive	1	0	0	0	0
20	positive	1	0	0	1	0

Picture 15. Representation of the data with the vectors and their sentiment being transformed as matrices during the creation of the models.

	0	1	2	3	4	5
0	1	1	1	1	1	0
1	1	0	0	0	0	1
2	1	0	0	1	0	0
3	1	0	0	0	0	0
4	1	0	0	0	0	0
5	1	0	0	0	0	0
6	1	0	0	0	0	0
7	1	0	0	0	0	0
8	1	0	0	0	0	0
9	1	0	0	0	0	0
10	1	0	0	0	0	0
11	1	0	0	1	0	0
12	1	0	0	0	0	0
13	1	0	0	0	1	0
14	1	0	0	0	0	0
15	1	0	0	0	0	0
16	1	0	0	0	0	0
17	1	0	0	0	0	0
18	1	0	0	0	0	0
19	1	0	0	0	0	0
20	1	0	0	1	0	0

Picture 16. Representation of the data containing the vectors being transformed as matrices during the creation of the models.

```
Table to test the prediction of the linear model and the real results:
Prediction linear model: 0 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 0 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
```

Picture 17. Representation of the result obtained during the testing of the prediction for the two models with the corpus "corpus-text-medium.txt".

```
Set = []
for i in data_message:
    a = pre_processing(i.lower())                                    #Pre
    b = nltk.FreqDist(a)                                             #Cal
    features = dict()
    word_features = list(b.keys())
    word_features_values = list(b.values())
    n = 0
    for i in word_features:
        features[i] = word_features_values[n]
        n += 1
    print("Features words of the tweet with the numbers it appears :",features)
    for i in word_features:
        if i in Set:
            continue
        else:
            Set.append(i)
```

Picture 18. Implementation of the pre-processing for the messages, and putting the features into a list.

```
Probability to get a good result for linear model: 0.8260869565217391
Probability to get a good result for rbf model: 0.8641304347826086
```

Picture 19. Results obtained for the accuracy of the two models for the corpus "corpus-text-medium.txt".

```
with open("Data-to-be-filled-no_glove.txt","w") as csv_
    fieldnames = ["Sentiment"]
    for i in Set:
        fieldnames.append(i)
    writer = csv.DictWriter(csv_file, fieldnames=fieldn
    writer.writeheader()
    writer = csv.writer(csv_file)
    n = 0
    for i in data_message:
        a = [data_sentiment[n]]
        for j in finaldict[n]:
            a.append(j)
        writer.writerow(a)
        n += 1
csv_file.close()
```

Picture 20. Implementation to write in a new file "Data-to-be-filled-no_glove.txt" the vectors of the messages and their sentiment.

```
data_message = [] #Create set in which we put the mess
data_sentiment = [] #Set in which we put the sentiment
with open("corpus-text-medium.txt") as csv_file:  #Rea
    csv_reader = csv.reader(csv_file, delimiter=",")
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            line_count += 1
            continue
        elif row == []:
            break
        else:
            data_message.append(row[4])#Add message#
            data_sentiment.append(row[1]) #Add sentime
    csv_file.close()
```

Picture 21. Implementation to get the messages and the sentiment of the corpus "corpus-text-medium.txt" and put them in two separated lists.

```
True positive probability for linear model: 0.9293478260869565
False positive probability for linear model: 0.07065217391304347
True negative probability for linear model: 0.8967391304347826
False negative probability for linear model: 0.10326086956521739

True positive probability for rbf model: 0.9619565217391304
False positive probability for rbf model: 0.03804347826086957
True negative probability for rbf model: 0.9891304347826086
False negative probability for rbf model: 0.010869565217391304
```

Picture 22. Results obtained from the table prediction of the two models with the corpus "corpus-text-medium.txt"

```
machine_learning_model_linear = "M_L_L_Model.pkl"
machine_learning_model_rbf = "M_L_R_Model.pkl"
joblib.dump(model,machine_learning_model_linear)
joblib.dump(model2,machine_learning_model_rbf)
```

Picture 23. Implementation to save the models created under a name.

	0	1	2	3	4	5
0	positive	positive	positive	positive	positive	positive

Picture 24. Representation of the inital_type label for the sentiment during the creation of the models.

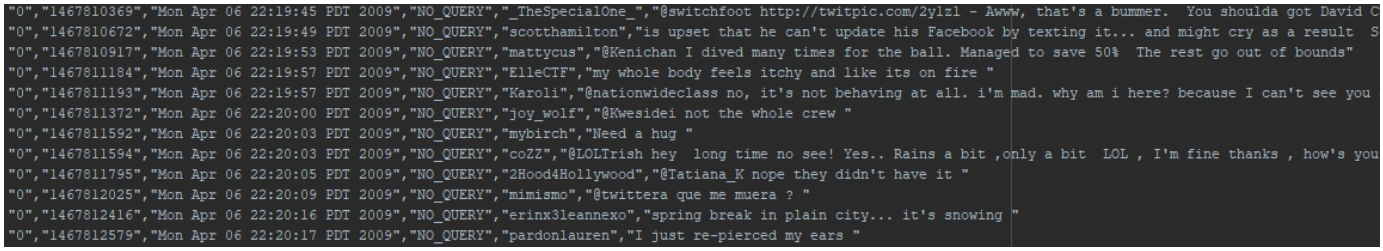
	0	1	2	3	4	5
0	0	0	0	0	0	0

Picture 25. Representation of the final_type label for the sentiment during the creation of the models.

```
import re,nltk

def pre_processing(message):
    #print("Pre-processing of the message:", message)
    dict2 = list()
    dict1 = nltk.sent_tokenize(message)  #Split the tweet in sentences#
    #print("Message tokenized by sentences:", dict1)
#########################################################
    for i in dict1:
        a = ' '.join(re.sub("(@)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", i).split())
        dict2.append(a)
    #print("Message without special characters:", dict2)
#########################################################
    dict1 = []
    for i in dict2:
        a = nltk.word_tokenize(i)                              #Split each sentence
        dict1.append(a)
    #print("Message tokenized as words:", dict1)
#################################################
    filter_words = []
    stop_words = set(nltk.corpus.stopwords.words("english"))
    for i in dict1:                                          # For each word, will
        for j in i:
            if j not in stop_words:
                filter_words.append(j)
    #print("Filtered words without English stopwords:", filter_words)
#################################################
    dict2 = []
    for i in filter_words:  # Lemmatization#
        a = nltk.stem.WordNetLemmatizer().lemmatize(i)
       # b = nltk.stem.PorterStemmer().stem(a)
        dict2.append(a)
    #print("Lemmatization of each word:", dict2)
#################################################
    return dict2
```

Picture 26. Implementation of the steps done during the pre-processing of a message.



Picture 27. Representation of the file of data from the corpus "corpus-1m600.txt".

```
possibletreats = []
sentiment= []
with open("corpus-text-1m600.txt") as csv_file:
    csv_reader = csv.reader(csv_file,delimiter=",")
    for row in csv_reader:
        if row == []:
            break
        else:
            for i in wordstocheck:
                if i in row[5].lower():
                    for j in range(wordstocheck.index(i)+1,len(wordstocheck)):
                        if wordstocheck[j] in row[5].lower():
                            if row[5] in possibletreats:  # The corpus might h
                                break
                            else:
                                possibletreats.append(row[5])
                                if row[0] == "4":
                                    sentiment.append(1)
                                else:
                                    sentiment.append(0)
                                break
                    break

    csv_file.close()
```

Picture 28. Implementation, to sort the messages having a possible terrorsim behaviour, and to put the messages and the sentiment in two different lists.

```
Lineset = []
for i in range(51):
    Lineset.append(i)

n = 0
with open("Data-to-be-filled-possible-terrorism-with_glove.txt", "w") as csv_file:
    writer = csv.writer(csv_file)
    writer.writerow(Lineset)
    for i in range(len(possibletreats)):
        x = [sentiment[n]]
        for j in Final_glOve_set[n]:
            x.append(j)
        writer.writerow(x)
        n += 1

csv_file.close()
```

Picture 29. Implementation to put in a new file "Data-to-be-filled-possible-terrorism-with_glove.txt" all the vectors of the messages with their sentiment

```
Set = []
a = 1
for i in range(50):
    Set.append(str(a))
    a += 1

newdataset = pd.read_csv("Data-to-be-filled-possible-terrorism-with_glove.txt")                    #Put in
newdataset.head()
features_words = newdataset[Set].as_matrix()
type_label = newdataset["0"].as_matrix()

print("Division of a set of vectors for training and testing.")
X_train, X_test, Y_train, Y_test = train_test_split(features_words,type_label, test_size = 0.3,random_state= 0)

print("Creation of the linear model and the rbf model.")
model = svm.SVC(kernel="linear", C=5, class_weight="balanced")                                  #model for kerne
model2 = svm.SVC(kernel="rbf", C=3, class_weight = "balanced", gamma = 0.01)                #mode for kernel = rbf

print("Input of the training vectors in each model.")
model.fit(X_train,Y_train)
model2.fit(X_train,Y_train)                                                    #Fit the models with the training da
```

Picture 30. Implementation, for the creation of the matrices for the vectors and sentiment from the file "Data-to-be-filled-no_glove.txt", for the set of training and testing vectors chosen randomly, the creation of the two models (linear and using the kernel trick, fitting of the two models with the training sets.

```
import pandas as pd

from time import sleep
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib
```

Picture 31. Representation of the libraries to be imported in Python 3 to use the Machine Learning with the Support Vector Machine.

```
Probability to get a good result for linear model: 0.6095679012345679
Probability to get a good result for rbf model: 0.6435185185185185
```

Picture 32. Results obtained by calculating the accuracy of the two models with the corpus "corpus-1m600.txt".

```
initialfeatures = []
for i in range(len(Set)):
    initialfeatures.append(0)
finalfeatures = initialfeatures.copy()

finaldict = []
m = 0
for i in data_message:
    m += 1
    n = -1
    initialfeatures = finalfeatures.copy()
    for j in Set:
        n += 1
        if j in i.lower():
            initialfeatures[n] += 1
    finaldict.append(initialfeatures) #Sho
```

Picture 33. Implementation to create the vectors of the messages based on the features.

```
True positive probability for linear model: 0.8873456790123457
False positive probability for linear model: 0.11265432098765432
True negative probability for linear model: 0.7222222222222222
False negative probability for linear model: 0.2777777777777778

True positive probability for rbf model: 0.9274691358024691
False positive probability for rbf model: 0.07253086419753087
True negative probability for rbf model: 0.9567901234567902
False negative probability for rbf model: 0.043209876543209874
```

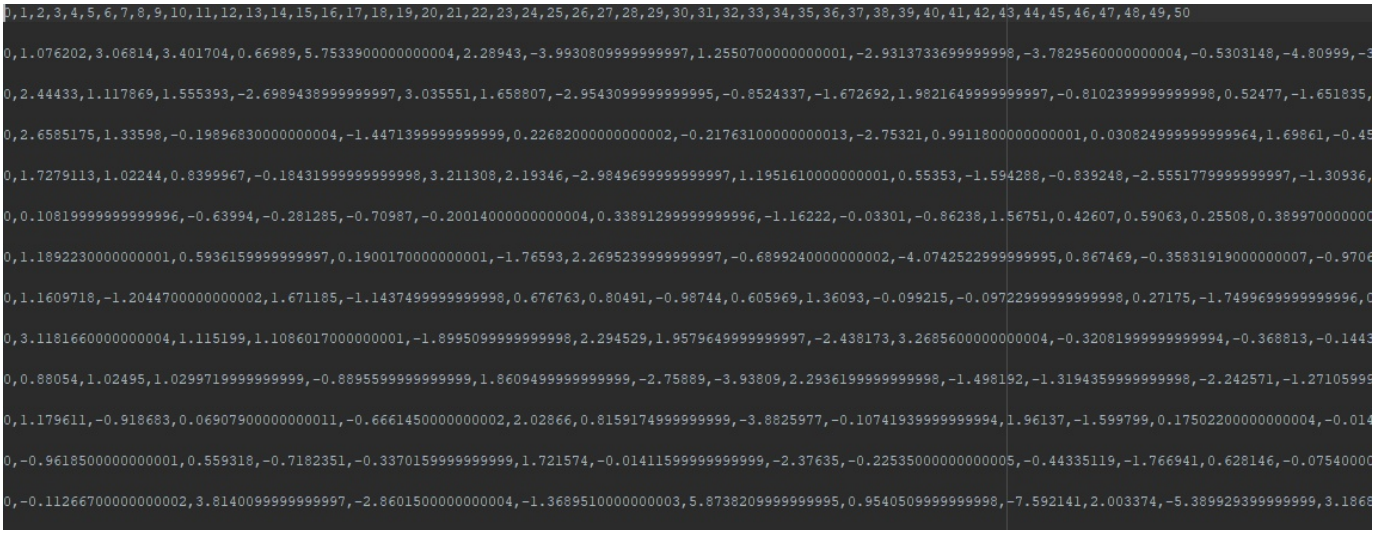
Picture 34. Results obtainedby calculating the table of prediction for the two models with the corpus "corpus-1m600.txt"

```
machine_learning_model_linear_new_domain = "M_L_L_N_D_Model.pkl"
machine_learning_model_rbf_new_domain = "M_L_R_N_D_Model.pkl"
joblib.dump(model, machine_learning_model_linear_new_domain)  #Sa
joblib.dump(model2, machine_learning_model_rbf_new_domain)
```

Picture 35. Implementation to save the models created under a name.

```
Set = []
Final_glOve_set = []
counting = 0
for i in possibletreats:
    counting += 1
    GlOve_set = [0] * 50
    a = pre_processing(i.lower())
    for j in a:
        with open("GloVe_library.txt") as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=' ')
            for row in csv_reader:
                if row[0] == j:
                    for k in range(50):
                        GlOve_set[k] += float(row[k+1])
                    break
        csv_file.close()
    Final_glOve_set.append(GlOve_set)
    for i in a:
        if i in Set:
            continue
        else:
            Set.append(i)
```

Picture 36. Implementation to use the Glove library to create the vectors for the messages based on their own features.



Picture 37. Representation of the file "Data-to-be-filled-with_glove.txt" containing the vectors of the messages created with Glove.

```
wordstocheck = ["bomb","terrorism","explosion","gun","fear","scary","death","shot","dead","terrorist","arab","arabics","die","weapon","destruction","beard","vest",
                "religion","allah","kill","killer"]
```

Picture 38. Implementation of the list of words related to terrorism behaviour.

```
independent -0.45357 -0.61985 0.064357 0.15554 0.17757 0.022016 -0.57475 0.1447 0.48198 -0.30307 0.37627 -0.57294 -0.16392 0.67122 -0.4424 -0.19306 0.61665 0.13443
kind 0.21671 0.24481 -0.71644 -0.20907 0.6638 0.20129 0.14668 -0.16793 -0.10682 0.73506 -0.31978 0.26202 -0.2456 0.22476 0.43025 0.14983 0.49864 0.39437 -0.016636 -
airport 1.5719 0.58655 0.16413 0.35197 -0.36115 -1.6337 -0.62805 0.11577 0.34076 -0.61184 0.41981 -0.57607 -0.30551 -0.56476 -0.067369 0.82933 -0.71416 0.9558 -1.40
paul -0.096465 1.1386 -0.25749 -0.99578 0.34209 0.22507 -1.3788 -0.39773 -0.30438 -0.073191 -0.011268 0.64175 -1.3504 -0.55745 0.57598 -0.58674 0.6033 -0.81784 -0.7
judge -0.41753 -0.77215 -0.77358 0.10329 1.0391 0.14551 0.52221 1.0225 0.054505 -0.22843 -0.45173 0.47498 -0.49818 -0.24674 1.2705 -0.1224 -0.13576 -1.3487 0.1996 -
internet 0.57121 -0.23423 1.5067 0.82537 -0.45765 -0.61598 -1.2452 -1.3567 0.92592 1.1069 -0.19654 0.06855 -0.17516 0.28349 0.43723 -0.03652 -0.94216 -0.1436 0.5045
movement -0.30695 -0.33417 -0.41905 -0.77643 0.13629 0.21405 -0.14562 -0.41073 0.010722 0.72117 0.83375 -0.92261 -0.60803 0.47743 -1.3782 -0.39002 0.48779 -0.52937
room 0.51518 0.80125 -0.13731 -0.472 1.0321 -0.75538 -0.58585 -0.10406 -0.22021 -0.38029 -0.82568 -0.1288 -0.059862 0.8529 0.54697 0.43243 -0.54769 0.35936 -0.14251
followed 0.063949 0.058864 -0.37989 0.047681 -0.14611 0.5127 -0.68928 0.17873 -0.43478 0.020686 -0.11248 -0.43566 -0.77011 -0.075069 0.41098 -0.60185 -0.63966 -0.78
original 0.36584 0.47422 -0.26492 -0.11875 0.17367 0.77899 -1.0685 -0.86926 -0.5622 0.17868 0.044906 0.5156 -0.085913 0.11807 0.43924 -0.047662 -0.27741 0.16102 -0.
angeles -0.17063 0.8223 -0.058367 0.83355 -0.28705 -0.60105 -1.6279 0.0021288 -0.30087 0.45756 -0.49159 -0.75213 -0.39453 0.49286 0.72132 -0.79924 -0.50742 -0.21675
italy 1.7704 -0.77758 -0.95302 0.329 0.040391 -0.086352 -0.096326 -0.14525 -0.85415 -0.091315 0.71825 -0.8378 -0.71724 -0.30078 1.2588 -0.72728 -0.26415 -0.17469 -0
` -0.085825 0.6135 0.069353 -0.46893 0.61648 -0.4357 0.33897 -0.60111 -0.21054 0.49083 -0.68084 1.0032 -0.58766 0.42168 0.96016 0.23724 0.36332 0.45626 -0.20216 0.1
data 0.53101 -0.55869 1.7674 0.44824 0.22341 -0.34559 -0.77679 -0.96117 1.1669 0.074279 0.8147 -0.059428 0.064599 0.0015176 0.099179 0.36602 -0.98724 -0.83913 0.159
comes 0.27471 0.47997 -0.23538 -0.021829 0.52225 0.12483 -0.17147 -0.016587 0.29799 0.3601 -0.36681 0.26492 -0.1034 0.066563 0.5982 0.3153 -0.04902 0.1524 0.050207
parties 0.3215 -0.24613 -0.46952 0.13853 0.1013 0.72639 -0.056441 0.17746 -1.154 -0.15891 -0.57427 -0.73113 -0.23344 0.74365 -0.072829 -0.010908 0.7259 -0.60433 0.8
nothing 0.34784 0.037246 -0.26159 -0.12038 0.69783 0.0097814 -0.053437 0.34871 -0.3859 0.5132 -0.4509 0.27192 -0.56817 -0.3929 1.1978 0.46452 0.40148 -0.059308 0.06
sea 1.3549 0.96469 -0.65964 0.3297 0.025316 -0.39186 -0.45001 0.089164 0.59136 -1.0542 0.25812 0.74352 1.2827 0.52466 -0.42253 0.11421 0.75232 0.33939 -2.3079 0.034
bring 0.36521 0.22801 0.19305 -0.29554 0.36734 -0.12537 -0.038514 0.27262 0.20583 0.38828 -0.26067 0.29052 -0.1073 0.016016 0.46221 0.30846 0.75655 -0.14722 0.27429
2012 -0.66285 0.77312 0.0097657 0.5739 -0.61578 0.16859 -1.1578 -0.35501 0.2507 -0.1666 0.76235 -0.24595 -0.52504 -0.10052 1.1672 -0.093335 -0.070071 0.31558 -1.289
annual -0.37791 1.0773 -0.66116 -0.11279 0.45041 -0.56139 -0.58576 -0.93223 1.4882 -0.06564 -0.44609 -0.98404 0.88088 -0.18864 1.1978 -0.67449 -0.02678 -0.012949 -1
officer 0.048403 -0.60703 0.19108 0.18007 1.2004 -0.38643 -0.34321 0.39655 0.1951 -2.0934 0.39803 0.72219 -1.3195 -0.25446 -0.12097 -0.65 -0.77295 0.78416 -0.68292
```

Picture 39. Representation of the Glove library file containing a fixed vector of length fifty for each English words.

```
Table to test the prediction of the linear model and the real results:
Prediction linear model: 0 / Prediction rbf model: 1 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 0 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 1 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 1
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 1 / Prediction rbf model: 1 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 0
Prediction linear model: 0 / Prediction rbf model: 0 / Real value: 1
Prediction linear model: 1 / Prediction rbf model: 0 / Real value: 0
```

Picture 40. Results obtained for the table of prediction for the two models with the corpus "corpus-1m600.txt".

```
score = model.score(X_test,Y_test)
score2 = model2.score(X_test,Y_test)
PredictionY = model.predict(X_test)
PredictionY2 = model2.predict(X_test)
```

Picture 41. Implementation to calculate the accuracy and the table of prediction for the models.

7.References

[1] https://en.wikipedia.org/wiki/Support_vector_machine
[2] https://en.wikipedia.org/wiki/Machine_learning
[3] http://scikit-learn.org/stable/modules/cross_validation.html
[4] https://stackabuse.com/scikit-learn-save-and-restore-models/
[5] https://appdividend.com/2018/07/23/prepare-dataset-for-machine- learning-in-python/
[6] https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
[7] https://medium.com/nlpython/sentiment-analysis-analysis-part-2-support-vector-machines-31f78baeee09
[8] https://stackoverflow.com/questions/8376691/how-to-remove-hashtag-user-link-of-a-tweet-using-regular-expression
[9] https://docs.python.org/2/library/re.html
[10] https://machinelearningmastery.com/basic-concepts-in-machine-learning/
[11] https://www.lemagit.fr/conseil/Machine-Learning-vs-Deep-Learning-un-avion-a-helices-et-un-avion-a-reaction
[12] https://stackoverflow.com/questions/15063936/csv-error-field-larger-than-field-limit-131072
[13] https://nlp.stanford.edu/projects/glove/
[14] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html