I- Definition of the problem

The aim of the homework was to classify messages into spam/ham classes, and then build a model in order to predict whether a message is a spam or ham.
The messages are stored in the messages.txt file. Each line begins with the class (ham or spam) followed by the corresponding message.
The text file contains exactly 5 000 lines, corresponding to **5 000 messages**.

Basically, we are facing a classification problem that we will solve using machine learning algorithms such as SVM and Naïve Bayes.

II- Methods

1) Generation of the data sets

First of all, the first step was to **generate the training set and testing set**. Therefore, we have created the function split_data that takes as parameters the first and last line. So, the function returns the data corresponding to all the messages between these lines.

At the end of this process, we had 2 data:

- Training set: line 0 to 3999 = 4 000 messages
- Testing set: line 4000 to 4999 = 1 000 messages

2) Generation of the class outputs

The second step was based on the class output. Indeed, as said in the previous part, each message was preceded by the class it belongs.
So, we needed to **extract the class of each message** and compute them into Y_train (class of the training set) and Y_test (class of the testing set).
This job is completed by our function get_Y() that returns a matrix with a number of lines equal to the vertical size of the training_set (for Y_train) and testing_set (for Y_test).
If the message[i] is a ham, then Y[i]= 0. If message[i] is a spam, then Y[i]=1.

It is interesting to note that our **Y_test has 874 classes as hams and 126 classes as spams**.

```
print(collections.Counter(Y_test))
```

```
Counter({0.0: 874, 1.0: 126})
```

3) Generation of the dictionary

In this third step, we focus on the training set. It is represented by our function make_dictionary, and comes to resolve an important classification's problem according to our situation. Indeed, as mentioned, we need to classify messages into binary classes. By talking about messages, we are actually meaning **words**.
So, we strongly need to determine **which words, that we can call key words, has the most important weight to determine the final class.**

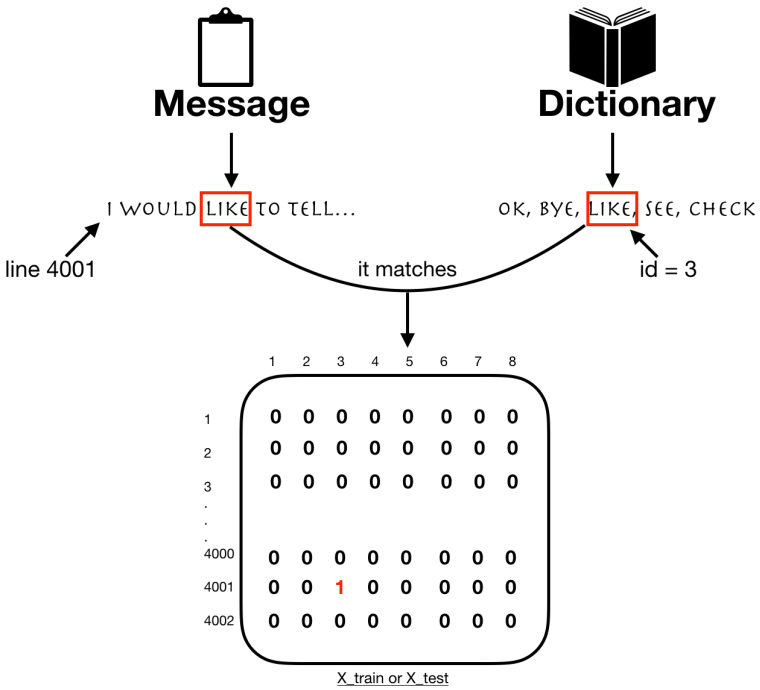
In our case, we firstly store all the words of the training set and process them **by ignoring the words "ham" and "spam"** that are actually the outputs and not the inputs. After that cleaning, we put the words in our dictionary with the word as key, and its number of occurrences as value.
Then, we clean once again our dictionary by deleting each word that is an **alpha type** (number…) or that has **only 1 character.** We can consider them as not relevant data.
Finally, we recreate the dictionary by adding only the **2500 most common words**. These 2500 words give us the solution to the previous problem: which words have the most important weight?

4) Extraction of the features

In this fourth step, we want to **extract the features** from the training set and testing set. Thanks to our extraction_features function, we create a matrix with a number of lines equal to the set we want to analyze, and number of columns equal to the length of our dictionary (most common words).
Each word has its own column, that is why number of columns = length of dictionary.
Each message corresponds to one line.

Basically, we initialize the matrix with value 0 everywhere, and then we scan both the message of each line and the dictionary in order to compare each word with the ones of the dictionary.
If the word appears in the dictionary, then we **replace** the 0 by the number of time the word appears.

The function returns a matrix that corresponds to the X_train or X_test, depending of the lines you pass in parameters.



5) Training SVM and Naive Bayes classifier and its variants

Now that we have our training data, training ouput, testing data and testing output, we can begin to classify.
Firstly, we need to create our models.

- Model 1 corresponds to the model we have built using SVM algorithm (LinearSVC)
- Model 2 corresponds to the model we have built using Naïve Bayes algorithm (MultinomialNB)

Using the fit function, we generate both of the models using de training data.

6) Testing phase

In order to validate our models, we need to test them by checking their accuracy.
Note that this time we only need to consider the testing data (testing inputs / outputs).
In order to do that, we have used 2 different solutions:

- **Score** function that displays the percentage of the accuracy
- **Confusion matrix** that is a technique for summarizing the performance of a classification algorithm.

III- Results

```
Score for training set with SVM : 0.999
Score for training set with Naive Bayes : 0.9835
```

Figure 2 : Score for training set

```
Score for testing set with SVM : 0.973
Score for testing set with Naive Bayes : 0.976
```

Figure 1 : Score for testing set

	Training set	Testing set
SVM	0.999	0.973
Naïve Bayes	0.9835	0.976

Figure 3 : Summary of the score

```
[[867   7]
 [ 20 106]]
```

Figure 4 : Confusion matrix with SVM

```
[[865   9]
 [ 15 111]]
```

Figure 5 : Confusion matrix with NB

Thanks to the confusion, that we will talk a little bit more in the next part, we can see that:

1) **With SVM**
- 867 messages predicted as hams are actually ham
- 106 messages predicted as spams are actually spams
- 20 messages predicted as spams are actually hams
- 7 messages predicted as hams are actually spams

2) **With Naive Bayes**
- 865 messages predicted as hams are actually hams
- 111 messages predicted as spams are actually spams
- 15 messages predicted as spams are actually hams
- 9 messages predicted as hams are actually spams

IV- Discussion of the results

The accuracy of both of our models is **very good**. Indeed, the accuracy for our test is about **97,3%** with SVM method, and **97,6%** with Naïve Bayes algorithm. Moreover, we succeeded to perform models that are **not neither over-fitting nor under-fitting**.
It implies that the model can **easily recogniz**e text messages as either spam or ham

This lab was the opportunity for us to use two different types of classifiers: **Discriminative classifiers** and **Generative classifiers**. Indeed, SVM belongs to discriminative classifiers whereas Naïve Bayes belongs to the generative one.
As mention in your course, in the websites *chioka.in* and *stackoverflow.com*:

- A generative classifier tries to **learn** the model that generates the data "behind the scenes" by estimating the assumptions and distributions of the model. Then, it uses this model to predict testing data because it assumes that the model that was learned captures the real model.
  For example, in our case, using Naïve Bayes method, the algorithm looks at the spam, and then can build a model of what a spam looks like. It does exactly the same of the ham by building another model. Finally, to classify a new message, it can match the new message against the spam model, and match it against the ham model, to see whether the new message looks more like the spam or more like the ham we had seen in the training set.
  In other words, it asks the question: based on my assumptions, which class **is most likely** to generate this message. So, this type of algorithm models how data was generated.
- A discriminative classifier has a different approach. Indeed, it tries to model by just depending on the observed data. We can say that it makes fewer assumptions on the assumptions on the distributions, but it actually depends heavily on the quality of the data.
  For example, in our case, using SVM method, the algorithm tries to find a hyperplane that is a decision boundary that separates the spam and ham. Then, to classify a new message as either a spam or a ham, it checks on **which side of the decision boundary it falls**, and makes predictions accordingly.

So, this type of algorithm does not care about of the data was generated, it simply categorizes a given message.

In the confusion matrix shown in figure 4 and 5, the diagonal elements show the correctly identified mails whereas non-diagonal elements represent wrong classification of mails.

We can see that Naïve Bayes **counts more corrected predictions** than SVM (976 vs 973 among 1000 instances) and therefore less errors.

So, in our case with our dataset, it seems that Naïve Bayes performs better than SVM, even if both are very close to each other (with a high accuracy).

In addition, we have decided to compare the execution's time of each algorithm using the library time and the function time(). The result shows that in our case**, Naïve Bayes is much faster than SVM**.

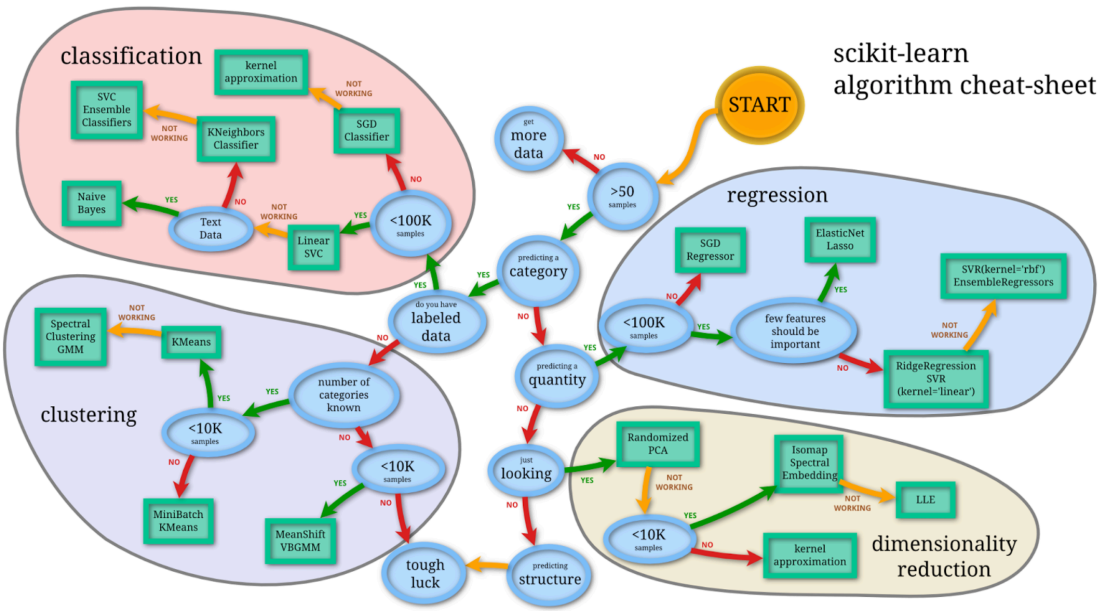
```
Time for SVM : 0.0597231388092041
Time for Naive Bayes : 0.025499820709228516
```

Note: We have tested the time with Jupyter, when kernels already existed. So, the time can be different. However, in general, Naïve Bayes is really faster than SVM

To conclude, we would like to underline that both of the algorithms can solve our initial problem: classify mails into ham/spam classes.

However, if we really need to choose one, we would select the **Naïve Bayes** algorithm because it has a very **good accuracy**, it is **faster** than SVM, it seems to be more appropriate since our **features are independent** and finally is mathematically **easier to understand**.

Some other applications will privilege a faster prediction than better accuracy. For example, credit card transaction, where it is more important to act quickly for fraud flag of transaction than 99% accuracy. 90% accuracy can be acceptable in this case.



Source http://scikit-learn.org

The image above shows that the corrected algorithm for our problem is Naïve Bayes since we **have a large dataset** (<100k samples) and a **binary classification** problem with **text data**.