**NNFS LAB 3: Bayes Classification**

**Bayes Classification**

Group 1:

Parth Deshpande 191060022

Ishan Deshpande 191060021

Dev Sharma 191060023

Kedar Daithankar 191060019

**Classifier:**

A classifier is a machine learning model that is used to discriminate different objects based on certain features.

**Principle of Naive Bayes Classifier:**

A **Naive Bayes** classifier is a probabilistic machine learning model that’s used for classification task. The crux of the classifier is based on the Bayes theorem.

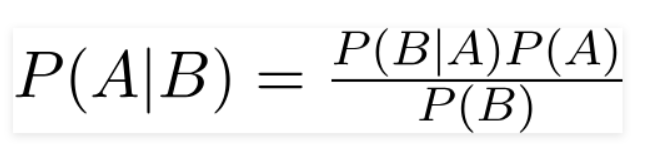
Naive Bayes classifier calculates the probabilities for every factor. Then it selects the outcome with highest probability.

This classifier assumes the **features are independent**. Hence the word naive.

The Bayes Classifier is a probabilistic model that makes the **most probable prediction** for a new example.

It is described using the Bayes Theorem that provides a principled way for calculating a conditional probability.

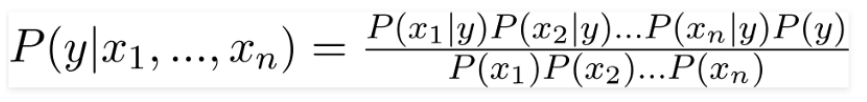
**Bayes Theorem:**



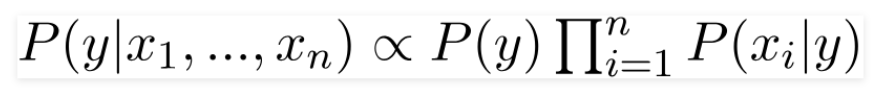
Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Now, let’s see the naïve bayes for multiple feature example

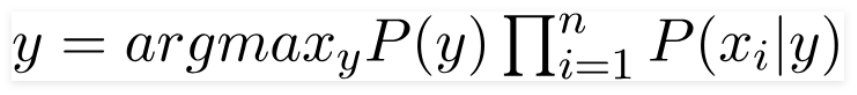
Here x1,x2….xn represent the features. By substituting for X and expanding using the chain rule we get,



Now, you can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change, it remains static. Therefore, the denominator can be removed and a proportionality can be introduced.



For binary predictions,



**Types of Naive Bayes Classifier:**

* Multinomial Naive Bayes:

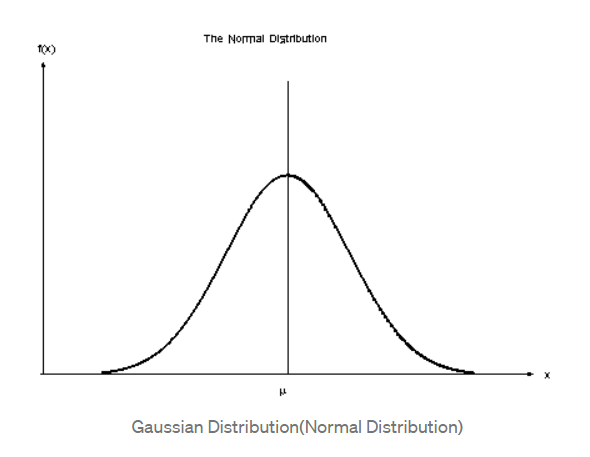
This is mostly used for document classification problem, i.e. whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

* Bernoulli Naive Bayes:

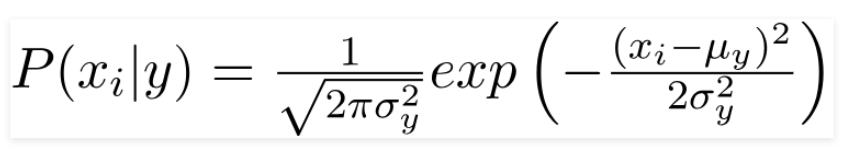
This is similar to the multinomial naive bayes but the predictors are Boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

* Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.



Since the way the values are present in the dataset changes, the formula for conditional probability changes to,



Implementation:

https://github.com/heisenberg-88/nnfs\_lab3\_bayes\_classification

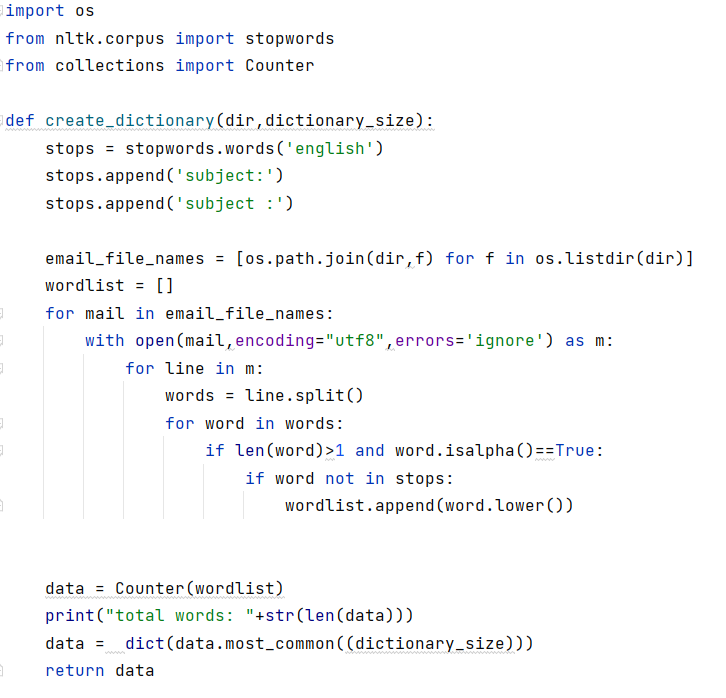
For this bayes classification we’ve used two datasets:

1. Ham-spam dataset
2. Enron-Spam dataset

(Source: <http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html> )

**Dictionary\_maker.py**

Here we’ve implemented function which creates dictionary {key : value, …} from all training mail text files.



**Returns: dictionary of {word: frequency}**

We’ve removed all the **stop-words** from the word set so that we get unbiased and true data.

Stop words are the words in a stop list which are filtered out before or after processing of natural language data because they are **insignificant.**

We have used **NLTK stop-words** list and ignored all words which are in this list as well as the characters which are punctuation marks, numeric values, dates, etc.

from nltk.corpus import stopwords

…

…

stops = stopwords.words('english')  
stops.append('subject:')  
stops.append('subject :')

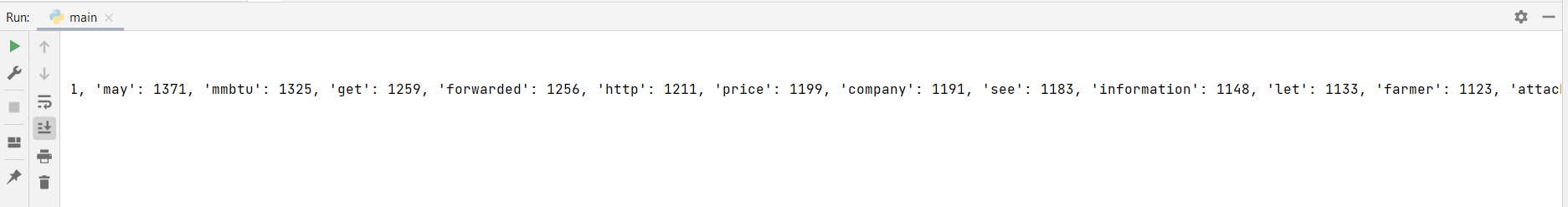
…

…

if len(word)>1 and word.isalpha()==True:  
 if word not in stops:  
 wordlist.append(word.lower())

This Function iterates through all files in the target folder and then it splits each line into words.

And then we create a dictionary of all these words which contain key value pairs like this, **{word : frequency}**

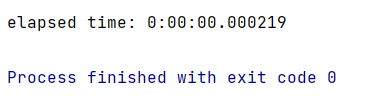


**Features\_extractor.py**

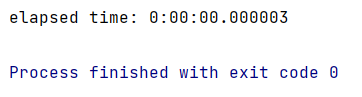
**(We’ve implemented this completely new approach for feature extraction which is faster than the one from article)**

This function creates features and labels array in minimal time. We’ve significantly reduced the execution time from elapsed time: 0:00:00.000**219** to 0:00:00.00000**3**

Algorithm from article:



Our optimized approach:



This new function optimally reduces time by creating dictionaries for each mail and then comparing this current dictionary to the main dictionary generated from training set of data.

features\_matrix = np.zeros((len(files),dictionary\_size))

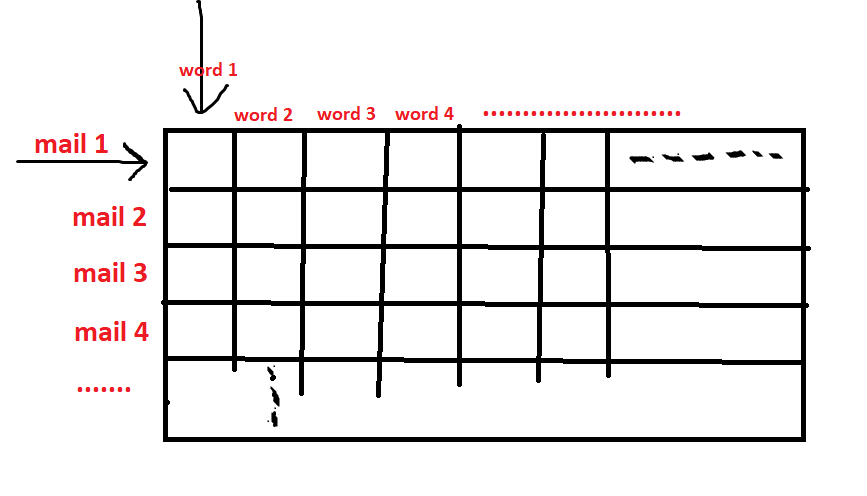
labels = np.zeros(len(files))

**Features matrix** is a matrix of dimensions

(no. of mails **x** no. of words in dictionary)

**Labels** is a 1d vector consisting of 1 or 0 according to weather the mail is spam or not.

(no. of mails **x** 1)

features\_matrix:

--------

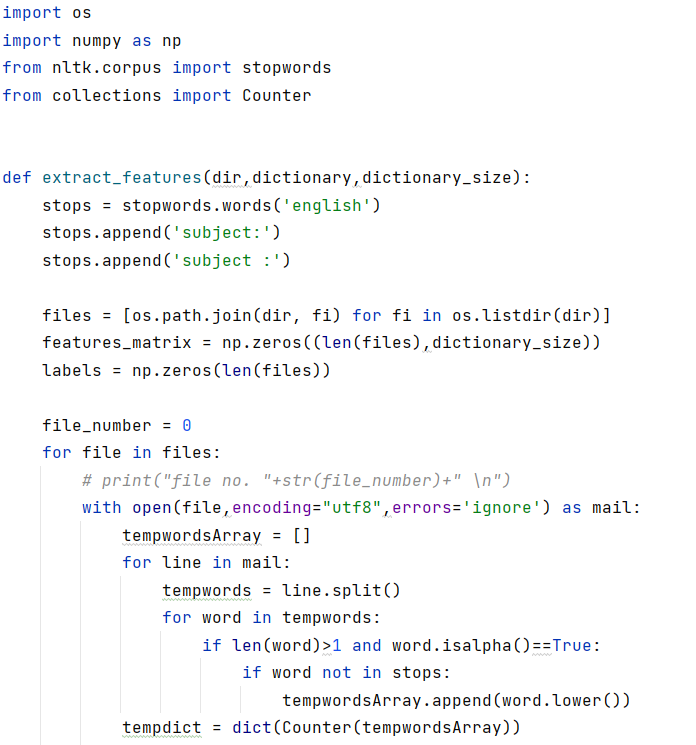
For optimization we’ve created **tempdict** (dictionary of words in that mail)

with open(file,encoding="utf8",errors='ignore') as mail:  
 tempwordsArray = []  
 for line in mail:  
 tempwords = line.split()  
 for word in tempwords:  
 if len(word)>1 and word.isalpha()==True:  
 if word not in stops:  
 tempwordsArray.append(word.lower())

# dictionary created below  
 tempdict = dict(Counter(tempwordsArray))

And then compared all words from main **dictionary**

index = 0  
for dictword in dictionary:  
 *# print("searching "+dictword+" in tempdict")* if dictword in tempdict:  
 *# print("found..")* features\_matrix[file\_number][index] = tempdict[dictword]  
 index+=1





**Main.py**

****

Here, required libraries are imported and the functions that we’ve created before are also imported.

**dictionary\_size** is used for setting the max number of words the dictionary can have.

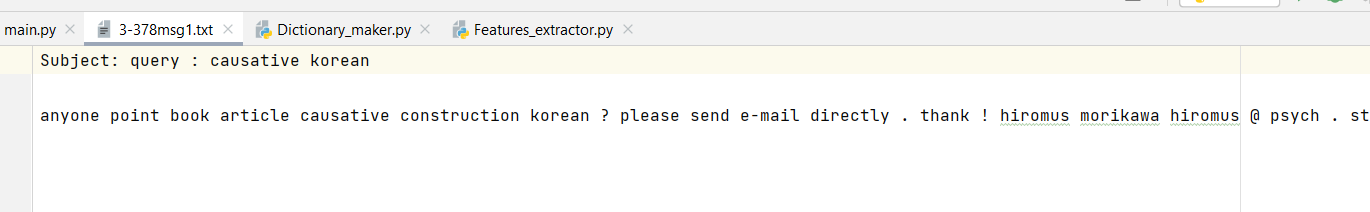
For calculating the elapsed time, time() is used.

import time  
from datetime import timedelta  
  
start = time.time()

……

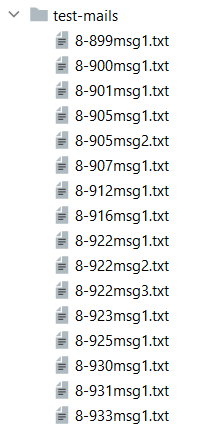
……

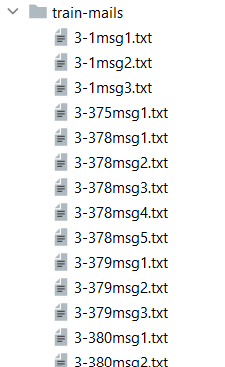
end = time.time()  
diff = end-start  
print("elapsed time: "+str(timedelta(microseconds=diff)))

We’ve used 2 types of datasets, let’s explore the first one which is given in the article provided.

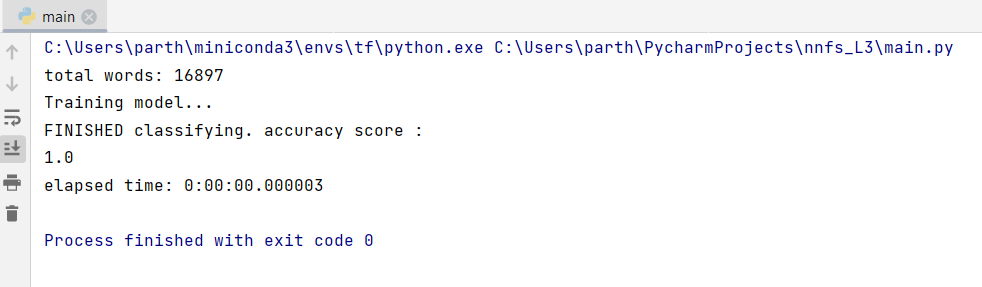
This is the sample mail from the spam-mail dataset.

All mails from this dataset have one line subject and one line body. Due to this, the dataset dictionary has a smaller number of significant words which contribute to the bayes classification and we get overfitted results.



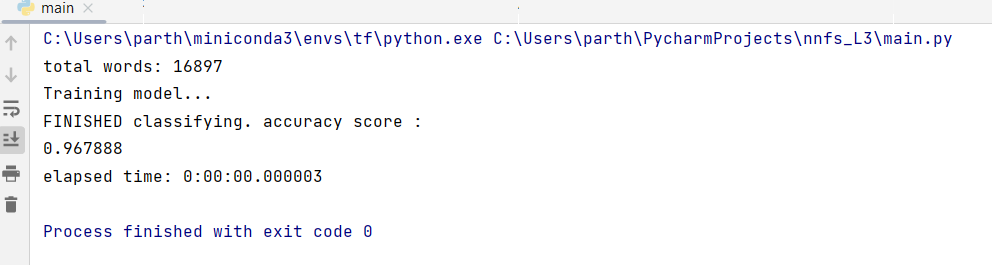


For **dictionary\_size** = 3000,

We get accuracy near to 100% which is not true in realistic situations.

Let’s try changing number of words in dictionary\_size,

Now, **dictionary\_size** = 7000



Here, we’re getting accuracy of around 97%

Similarly, we can try this for **GaussianNB,MultinomialNB,BernoulliNB**

**By using model = GaussianNB()**

**model = MultinomialNB()**

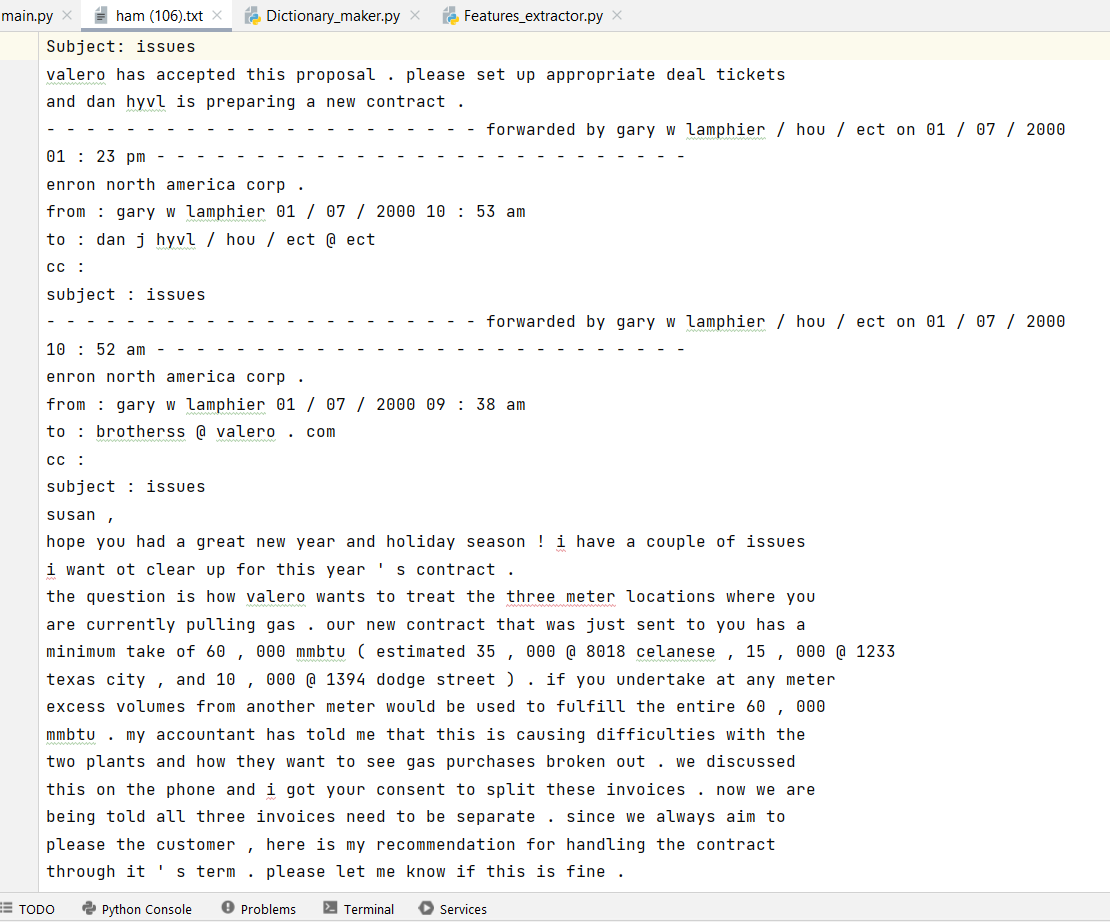
**model = BernoulliNB()**

we got nearly same accuracy for these three models

Let’s explore the second dataset.

**Enron-Spam dataset**

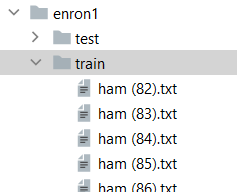
(Source: <http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html> )

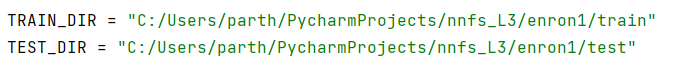
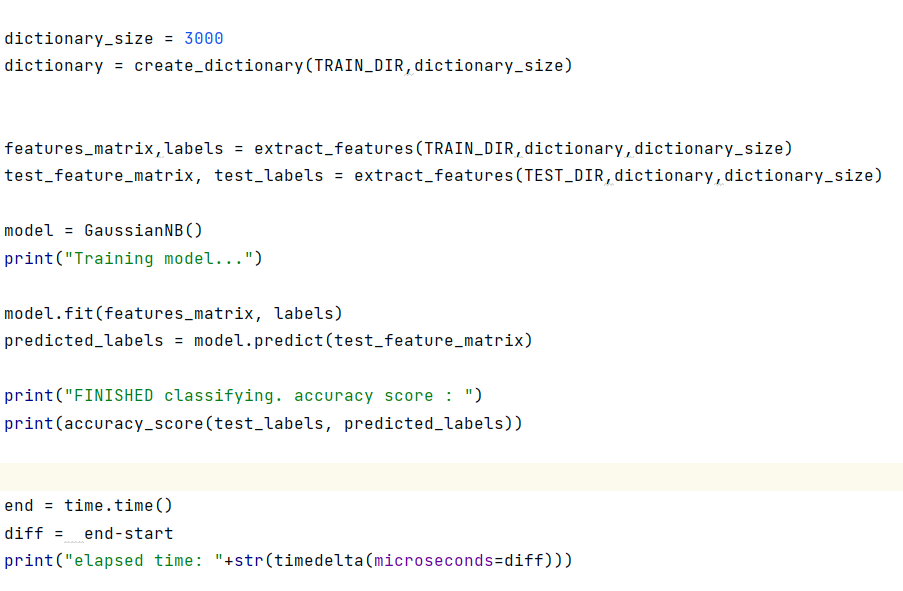


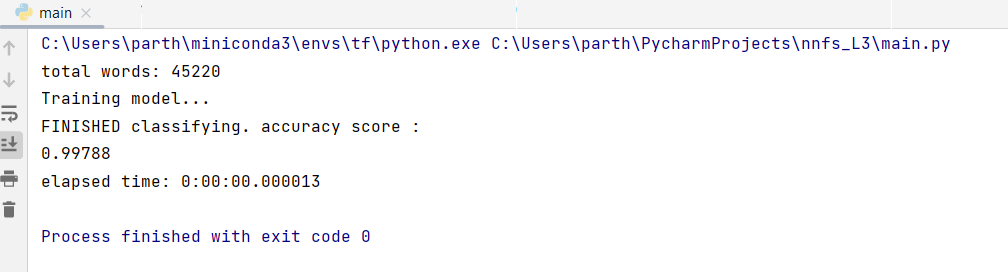
We can clearly see that emails from this dataset have large email body.

So that we can get a greater number of words in our dictionary and the classifier will work better.

We’ve also made this dataset compatible with the algorithm.

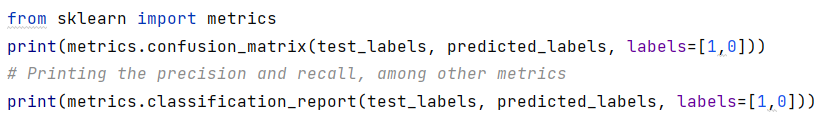


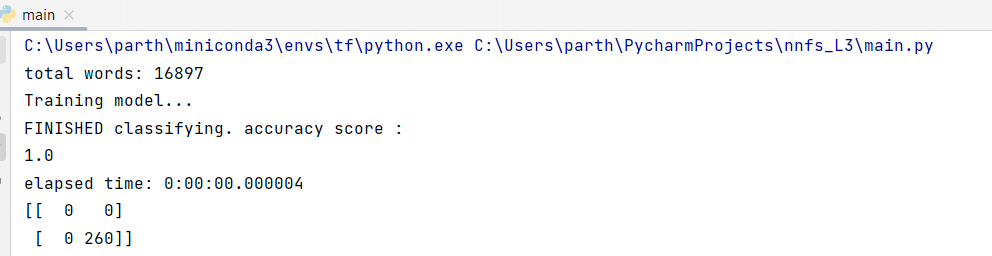


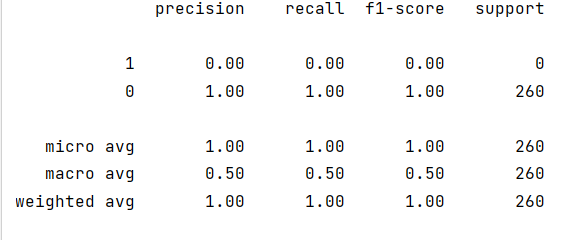


As the total number of words in this dataset is ~45000, we can get better classification if we increase number of words in dictionary.

For smaller dataset,

We can generate this confusion matrix and other metrics:





**Conclusion:**

* Naive Bayes algorithms are mostly used in spam filtering, recommendation systems etc.
* They are fast and easy to implement.
* Biggest disadvantage is that the requirement of predictors to be independent. In most of the real-life cases, the predictors are dependent, this hinders the performance of the classifier.