****

Machine Learning

B9DA104

|  |  |
| --- | --- |
| Student Names | Student ID number (email) |
| 1-Brijith Elizabeth Joseph | [10393634@mydbs.ie](mailto:10393634@mydbs.ie) |
| 2-Jayaramakrishnan Balakrishnan | [10393364@mydbs.ie](mailto:10393364@mydbs.ie) |

**Assignment : CA2**

**Naive Bayes Algorithm**

**Brijith Elizabeth Joseph**

**10393634**

**The Naive Bayes algorithm**

The Naive Bayes algorithm describes a simple method to apply Bayes' theorem to classification problems. Although it is not the only machine learning method that utilizes Bayesian methods, it is the most common one.The Naive Bayes algorithm is named as such because it makes some "naive“ assumptions about the data. In particular, Naive Bayes assumes that all of the features in the dataset are equally important and independent. These assumptions are rarely true in most real-world applications.

Bayesian classifiers are best applied to problems in which the information from numerous attributes should be considered simultaneously in order to estimate the overall probability of an outcome. many machine learning algorithms ignore features that have weak effects, Bayesian methods utilize all the available evidence to subtly change the predictions. If large number of features have relatively minor effects, taken together, their combined impact could be quite large. Bayesian probability theory is rooted in the idea that the estimated likelihood of an event, or a potential outcome, should be based on the evidence at hand across multiple trials, or opportunities for the event to occur. The probability of all the possible outcomes of a trial must always sum to 1, because a trial always results in some outcome happening. Thus, if the trial has two outcomes that cannot occur simultaneously.we are interested in monitoring several non mutually exclusive events for the same trial. If certain events occur with the event of interest, we may be able to use them to make predictions

They have been used successfully for:

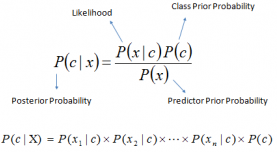
• Text classification, such as junk e-mail (spam) filtering

• Intrusion or anomaly detection in computer networks

• Diagnosing medical conditions given a set of observed symptoms

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).



* P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
* P(c) is the prior probability of class.
* P(x|c) is the likelihood which is the probability of predictor given class.
* P(x) is the prior probability of predictor.

***Pros:***

* It is easy and fast to predict class of test data set. It also perform well in multi class prediction
* When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
* It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

***Cons:***

* If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
* On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict\_proba are not to be taken too seriously.
* Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

**Dataset**

The dataset ‘Spam Filter ’is took from Kaggle. It is an open dataset with 5728 records.The file is in the form of excel. It has two fields’ text and spam. The field text contains the message, and Spam contains binary data. 1 represents the message is spam and 0 represents the message is ham.

A spam filter is a program that is used to detect unsolicited and unwanted email and prevent those messages from getting to a user's inbox. Like other types of filtering programs, a spam filter looks for certain criteria on which it bases judgments. For example, the simplest and earliest versions (such as the one available with Microsoft's Hotmail) can be set to watch for particular words in the subject line of messages and to exclude these from the user's inbox. This method is not especially effective, too often omitting perfectly legitimate messages (these are called false positives) and letting actual spam through. More sophisticated programs, such as Bayesian filters or other heuristic filters, attempt to identify spam through suspicious word patterns or word frequency.

The url is : <https://www.kaggle.com/karthickveerakumar/spam-filter>

Using the dataset, we created a model that can predict whether a message is spam or ham. As per the content in the message, we are classifying the messages into two categories, Spam and Ham

**Code with explanation**

**Step 1: importing required modules**

import numpy as np **# linear algebra**

import pandas as pd # **data processing, CSV file I/O**

import matplotlib.pyplot as plt **# For plots and graph**

import seaborn as sns **# For plots and graph**

import re **# For manipulating regular expressions**

from wordcloud import WordCloud **# for displaying world cloud**

import nltk **#Natural Language Toolkit**

from nltk import word\_tokenize,sent\_tokenize

nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('punkt')

from sklearn.feature\_extraction.text import CountVectorizer

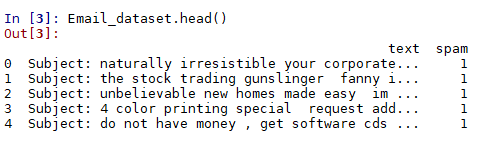
import warnings

warnings.filterwarnings('ignore')

**# Step 2: importing the Email\_dataset which is in csv format**

Email\_dataset = pd.read\_csv('Dataset for Naive Bayes.csv', encoding='latin-1')

Email\_dataset.head()



**# count observations in each label and display the distribution of spam using beautiful seaborn package**

label\_counts = Email\_dataset.spam.value\_counts()

plt.figure(figsize = (12,6))

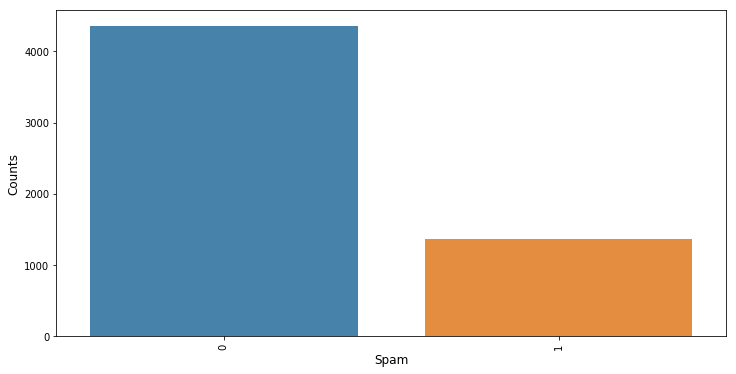
sns.barplot(label\_counts.index, label\_counts.values, alpha = 0.9)

plt.xticks(rotation = 'vertical')

plt.xlabel('Spam', fontsize =12)

plt.ylabel('Counts', fontsize = 12)

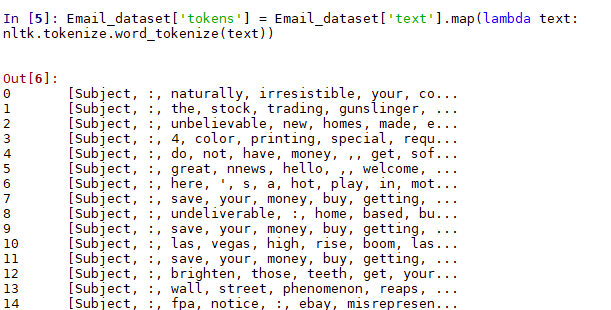
plt.show()



**# Step 3. Text Data preprocessing**

**Tokenization converts continuous stream of words into separate token for each word.**

Email\_dataset['tokens'] = Email\_dataset['text'].map(lambda text: nltk.tokenize.word\_tokenize(text))

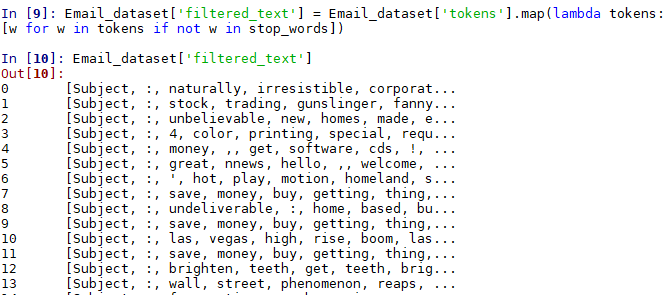


**#Removing stop words**

**Stop words usually refers to the most common words in a language like 'the', 'a', 'as' etc. These words usually do not convey any useful information needed for spam filter so its removed**

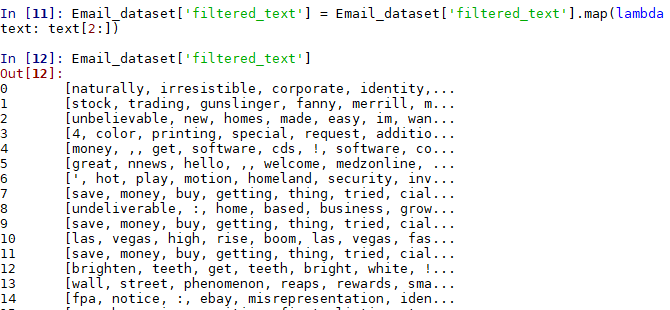
stop\_words = set(nltk.corpus.stopwords.words('english'))

Email\_dataset['filtered\_text'] = Email\_dataset['tokens'].map(lambda tokens: [w for w in tokens if not w in stop\_words])



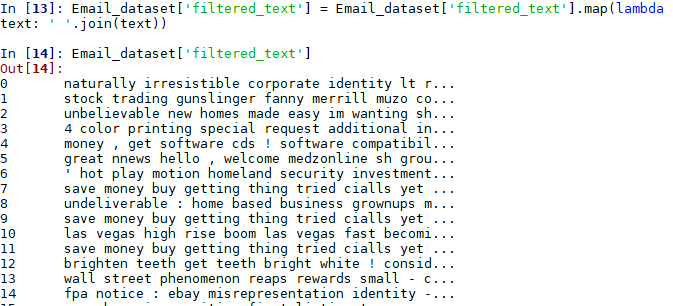
**#Every mail starts with 'Subject :' it is removed from each mail**

Email\_dataset['filtered\_text'] = Email\_dataset['filtered\_text'].map(lambda text: text[2:])



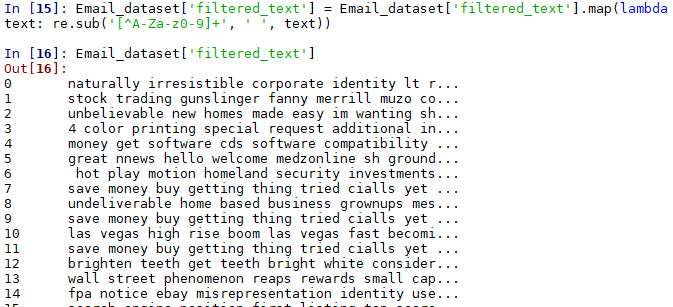
**#merge all tokens together in a string**

Email\_dataset['filtered\_text'] = Email\_dataset['filtered\_text'].map(lambda text: ' '.join(text))



**#Mails still have many special character tokens which may not be relevant for spam filter**

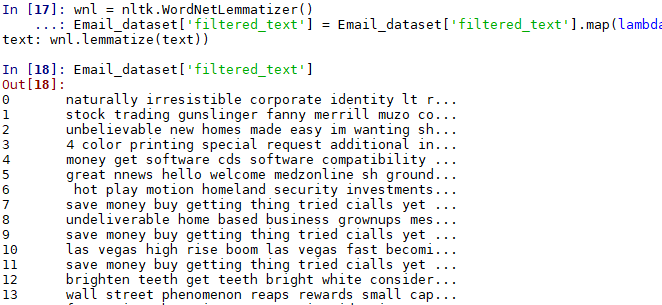
Email\_dataset['filtered\_text'] = Email\_dataset['filtered\_text'].map(lambda text: re.sub('[^A-Za-z0-9]+', ' ', text))



**#Lemmatization of the words : the process of grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the word's lemma, or dictionary form**

wnl = nltk.WordNetLemmatizer()

Email\_dataset['filtered\_text'] = Email\_dataset['filtered\_text'].map(lambda text: wnl.lemmatize(text))



**# Step 4:for splitting Email\_dataset into train set and test set in 4:1 ratio**

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(Email\_dataset["text"],Email\_dataset["spam"], test\_size = 0.2, random\_state = 10)

**# for vectorizing words**

vect = CountVectorizer(stop\_words='english')

vect.fit(X\_train)

X\_train\_df = vect.transform(X\_train)

X\_test\_df = vect.transform(X\_test)

**#step:5.Applying Machine Learning model**

prediction = dict()

**# Naive Bayes Machine Learning Model**

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

model.fit(X\_train\_df,y\_train)

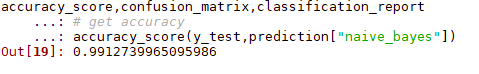
prediction["naive\_bayes"] = model.predict(X\_test\_df)

**# Step 6. Check performance of the model**

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

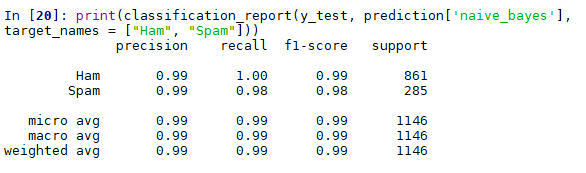
**# Accuracy**

accuracy\_score(y\_test,prediction["naive\_bayes"])



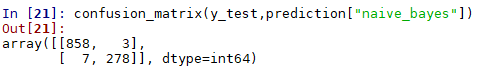
**# classification\_report**

print(classification\_report(y\_test, prediction['naive\_bayes'], target\_names = ["Ham", "Spam"]))



**#confusion\_matrix**

confusion\_matrix(y\_test,prediction["naive\_bayes"])

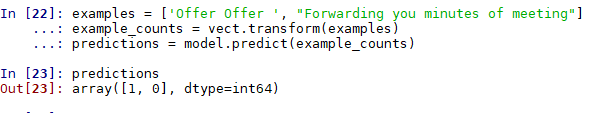


**#Predictions on sample text**

examples = ['great offer ', "Forwarding you minutes of meeting"]

example\_counts = vect.transform(examples)

predictions = model.predict(example\_counts)



statement\_testing = input ("Enter a message for testing")

statement\_testing=[statement\_testing]

example\_counts = vect.transform(statement\_testing)

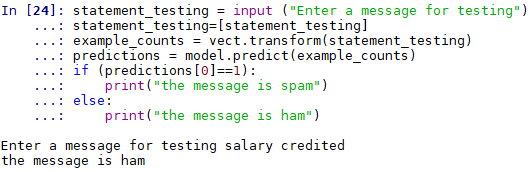
predictions = model.predict(example\_counts)

if (predictions[0]==1):

print("the message is spam")

else:

print("the message is ham")



#Wordcloud of spam mails

spam\_words = ''.join(list(Email\_dataset[Email\_dataset['spam']==1]['filtered\_text']))

spam\_wordclod = WordCloud(width = 512,height = 512).generate(spam\_words)

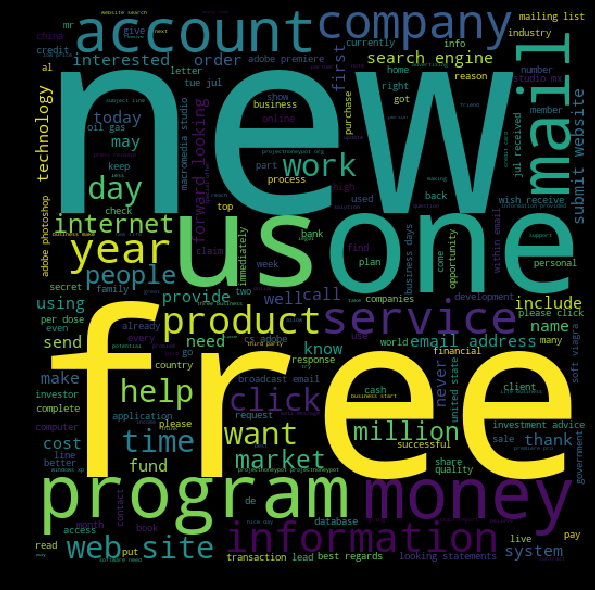
plt.figure(figsize = (10, 8), facecolor = 'k')

plt.imshow(spam\_wordclod)

plt.axis('off')

plt.tight\_layout(pad = 0)

plt.show()



#Wordcloud of non-spam mails

spam\_words = ''.join(list(Email\_dataset[Email\_dataset['spam']==0]['filtered\_text']))

spam\_wordclod = WordCloud(width = 512,height = 512).generate(spam\_words)

plt.figure(figsize = (10, 8), facecolor = 'k')

plt.imshow(spam\_wordclod)

plt.axis('off')

plt.tight\_layout(pad = 0)

plt.show()

