

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:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

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 x_1, x_2, \dots, x_n represent the features. By substituting for X and expanding using the chain rule we get,

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)}$$

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.

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y)$$

For binary predictions,

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

Types of Naive Bayes Classifier:

- Multinomial Naive Bayes:

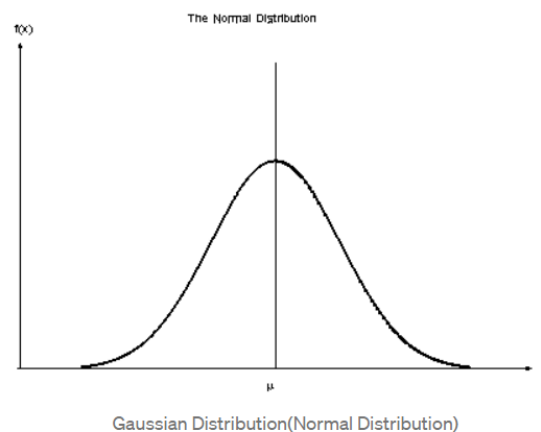
This is mostly used for document classification problem, i.e. whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

- 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,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp \left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2} \right)$$

Implementation:

https://github.com/heisenberg-88/nfbs_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.

```
import os
from nltk.corpus import stopwords
from collections import Counter

def create_dictionary(dir,dictionary_size):
    stops = stopwords.words('english')
    stops.append('subject:')
    stops.append('subject :')

    email_file_names = [os.path.join(dir,f) for f in os.listdir(dir)]
    wordlist = []

    for mail in email_file_names:
        with open(mail,encoding="utf8",errors='ignore') as m:
            for line in m:
                words = line.split()
                for word in words:
                    if len(word)>1 and word.isalpha()==True:
                        if word not in stops:
                            wordlist.append(word.lower())

    data = Counter(wordlist)
    print("total words: "+str(len(data)))
    data = dict(data.most_common((dictionary_size)))
    return data
```

Returns: dictionary of {word: frequency}

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

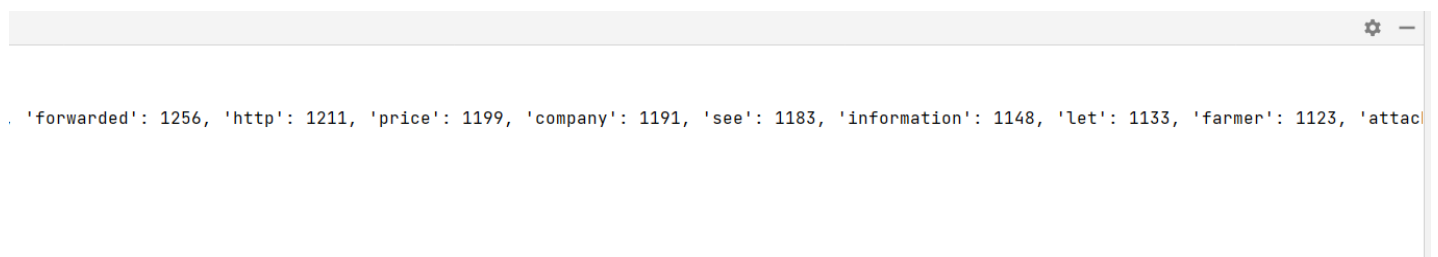
Stop words are the words in a stop list which are filtered out before or after processing of natural language data because they are **insignificant**.

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:

0:00:00.000219 to **0:00:00.000003**

Algorithm from article:

```
elapsed time: 0:00:00.000219

Process finished with exit code 0
```

Our optimized approach:

```
elapsed time: 0:00:00.000003

Process finished with exit code 0
```

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))

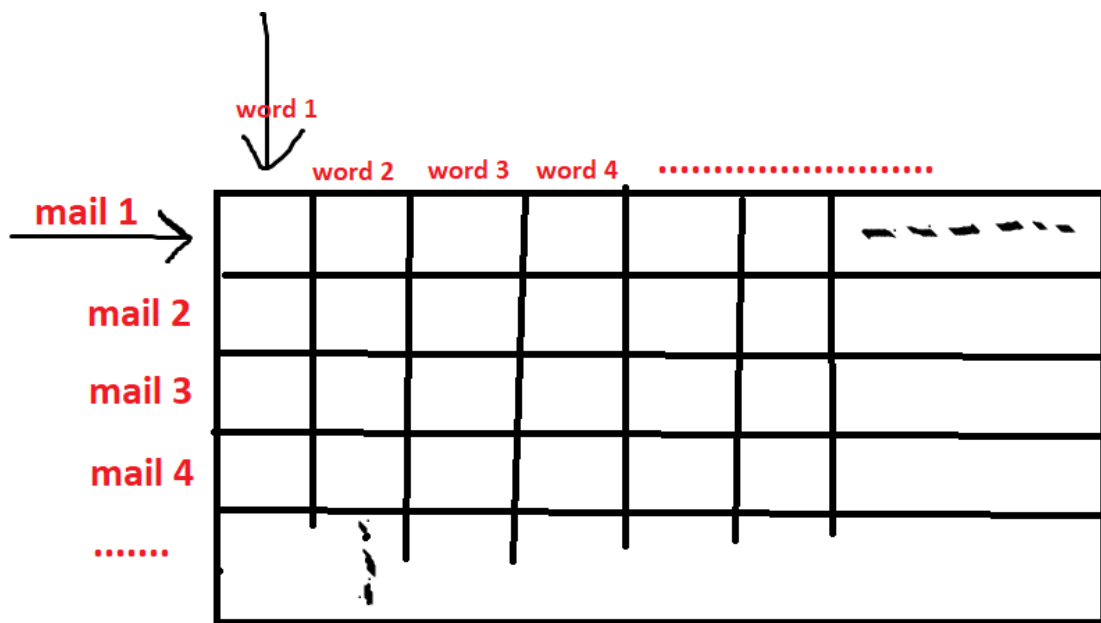
labels = np.zeros(len(files))
```

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 whether 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
```

```

import os
import numpy as np
from nltk.corpus import stopwords
from collections import Counter

def extract_features(dir,dictionary,dictionary_size):
    stops = stopwords.words('english')
    stops.append('subject:')
    stops.append('subject :')

    files = [os.path.join(dir, fi) for fi in os.listdir(dir)]
    features_matrix = np.zeros((len(files),dictionary_size))
    labels = np.zeros(len(files))

    file_number = 0
    for file in files:
        # print("file no. "+str(file_number)+" \n")
        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())
            tempdict = dict(Counter(tempwordsArray))

            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

            filepathwords = file.split('/')
            lastwordfrompath = filepathwords[len(filepathwords) - 1]
            if lastwordfrompath.startswith("spmsg"):
                labels[file_number] = 1

            file_number += 1

    return features_matrix , labels

```


Main.py

```
from Dictionary_maker import create_dictionary
from Features_extractor import extract_features
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score

TRAIN_DIR = "C:/Users/parth/PycharmProjects/nnfs_L3/train-mails"
TEST_DIR = "C:/Users/parth/PycharmProjects/nnfs_L3/test-mails"

# TRAIN_DIR = "C:/Users/parth/PycharmProjects/nnfs_L3/enron1/train"
# TEST_DIR = "C:/Users/parth/PycharmProjects/nnfs_L3/enron1/test"

dictionary_size = 3000
dictionary = create_dictionary(TRAIN_DIR, dictionary_size)

features_matrix, labels = extract_features(TRAIN_DIR, dictionary, dictionary_size)
test_feature_matrix, test_labels = extract_features(TEST_DIR, dictionary, dictionary_size)

model = GaussianNB()
print("Training model...")

model.fit(features_matrix, labels)
predicted_labels = model.predict(test_feature_matrix)

print("FINISHED classifying. accuracy score : ")
print(accuracy_score(test_labels, predicted_labels))
```

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

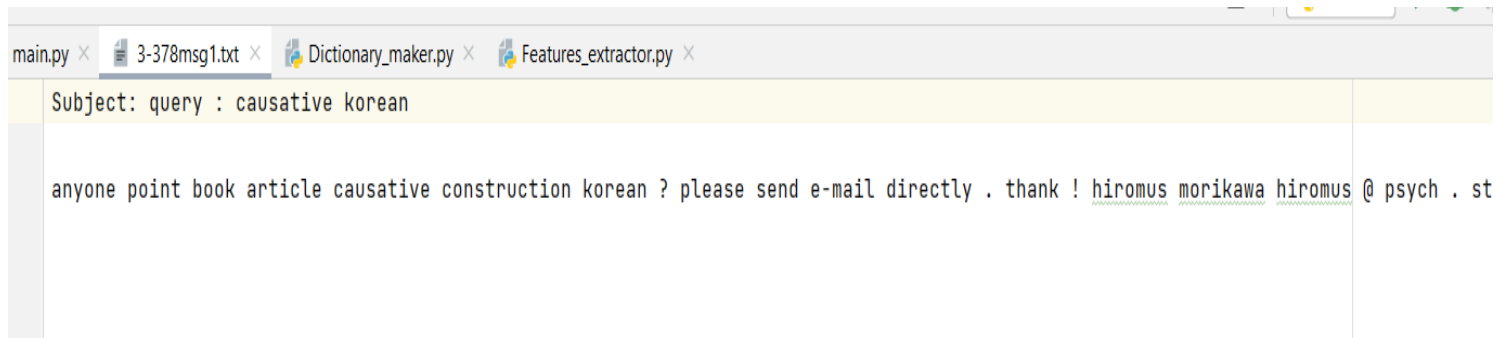
start = time.time()

.....

.....

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

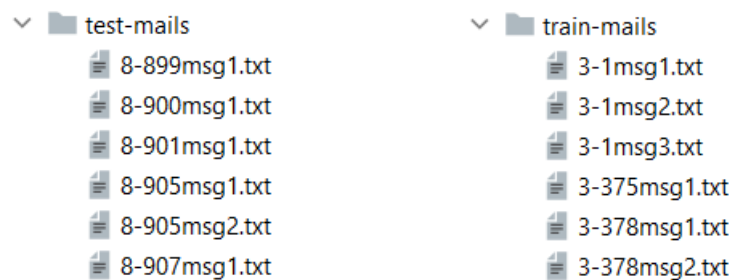
We've used 2 types of datasets, let's explore the first one which is given in the article provided.



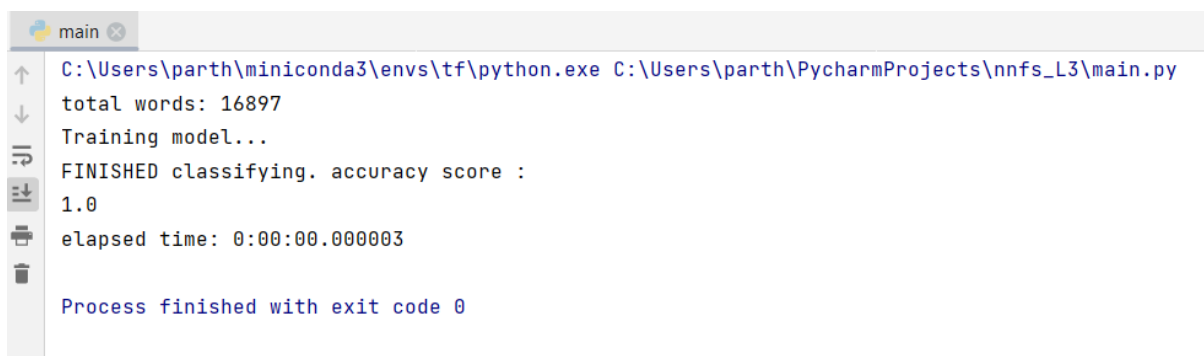
The screenshot shows a web browser with four tabs: 'main.py', '3-378msg1.txt', 'Dictionary_maker.py', and 'Features_extractor.py'. The active tab is '3-378msg1.txt', which displays an email. The subject line is 'Subject: query : causative korean'. The body of the email is 'anyone point book article causative construction korean ? please send e-mail directly . thank ! hiromus morikawa hiromus @ psych . st'. The email body is highlighted in yellow.

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.



The screenshot shows a terminal window with the following output:

```
C:\Users\parth\miniconda3\envs\tf\python.exe C:\Users\parth\PycharmProjects\nnfs_L3\main.py
total words: 16897
Training model...
FINISHED classifying. accuracy score :
1.0
elapsed time: 0:00:00.000003

Process finished with exit code 0
```

Let's try changing number of words in dictionary_size,
Now, **dictionary_size = 7000**

```
main x
C:\Users\parth\miniconda3\envs\tf\python.exe C:\Users\parth\PycharmProjects\nnfs_L3\main.py
total words: 16897
Training model...
FINISHED classifying. accuracy score :
0.967888
elapsed time: 0:00:00.000003

Process finished with exit code 0
```

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

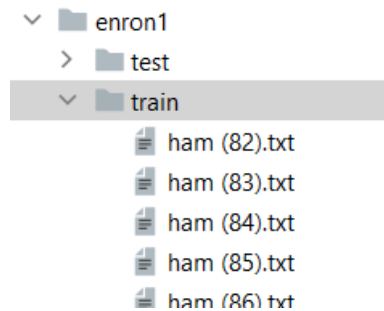
(Source: http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html)

```
main.py x ham (106).txt x Dictionary_maker.py x Features_extractor.py x
Subject: issues
valero has accepted this proposal . please set up appropriate deal tickets
and dan hyvl is preparing a new contract .
- - - - - forwarded by gary w lamphier / hou / ect on 01 / 07 / 2000
01 : 23 pm - - - - -
enron north america corp .
from : gary w lamphier 01 / 07 / 2000 10 : 53 am
to : dan j hyvl / hou / ect @ ect
cc :
subject : issues
- - - - - forwarded by gary w lamphier / hou / ect on 01 / 07 / 2000
10 : 52 am - - - - -
enron north america corp .
from : gary w lamphier 01 / 07 / 2000 09 : 38 am
to : brotherss @ valero . com
cc :
subject : issues
susan ,
hope you had a great new year and holiday season ! i have a couple of issues
i want ot clear up for this year ' s contract .
the question is how valero wants to treat the three meter locations where you
are currently pulling gas . our new contract that was just sent to you has a
minimum take of 60 , 000 mmbtu ( estimated 35 , 000 @ 8018 celanese , 15 , 000 @ 1233
texas city , and 10 , 000 @ 1394 dodge street ) . if you undertake at any meter
excess volumes from another meter would be used to fulfill the entire 60 , 000
mmbtu . my accountant has told me that this is causing difficulties with the
two plants and how they want to see gas purchases broken out . we discussed
this on the phone and i got your consent to split these invoices . now we are
being told all three invoices need to be separate . since we always aim to
please the customer , here is my recommendation for handling the contract
through it ' s term . please let me know if this is fine .
```

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.



```
TRAIN_DIR = "C:/Users/parth/PycharmProjects/nnfs_L3/enron1/train"
```

```
TEST_DIR = "C:/Users/parth/PycharmProjects/nnfs_L3/enron1/test"
```

```
dictionary_size = 3000
```

```
dictionary = create_dictionary(TRAIN_DIR,dictionary_size)
```

```
features_matrix,labels = extract_features(TRAIN_DIR,dictionary,dictionary_size)
```

```
test_feature_matrix, test_labels = extract_features(TEST_DIR,dictionary,dictionary_size)
```

```
model = GaussianNB()
```

```
print("Training model...")
```

```
model.fit(features_matrix, labels)
```

```
predicted_labels = model.predict(test_feature_matrix)
```

```
print("FINISHED classifying. accuracy score : ")
```

```
print(accuracy_score(test_labels, predicted_labels))
```

```
end = time.time()
```

```
diff = end-start
```

```
print("elapsed time: "+str(timedelta(microseconds=diff)))
```

```
main x
C:\Users\parth\miniconda3\envs\tf\python.exe C:\Users\parth\PycharmProjects\nnfs_L3\main.py
total words: 45220
Training model...
FINISHED classifying. accuracy score :
0.99788
elapsed time: 0:00:00.000013

Process finished with exit code 0
```

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:

```
from sklearn import metrics
print(metrics.confusion_matrix(test_labels, predicted_labels, labels=[1,0]))
# Printing the precision and recall, among other metrics
print(metrics.classification_report(test_labels, predicted_labels, labels=[1,0]))
```

```
main x
C:\Users\parth\miniconda3\envs\tf\python.exe C:\Users\parth\PycharmProjects\nnfs_L3\main.py
total words: 16897
Training model...
FINISHED classifying. accuracy score :
1.0
elapsed time: 0:00:00.000004
[[ 0  0]
 [ 0 260]]
```

	precision	recall	f1-score	support
1	0.00	0.00	0.00	0
0	1.00	1.00	1.00	260
micro avg	1.00	1.00	1.00	260
macro avg	0.50	0.50	0.50	260
weighted avg	1.00	1.00	1.00	260

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.