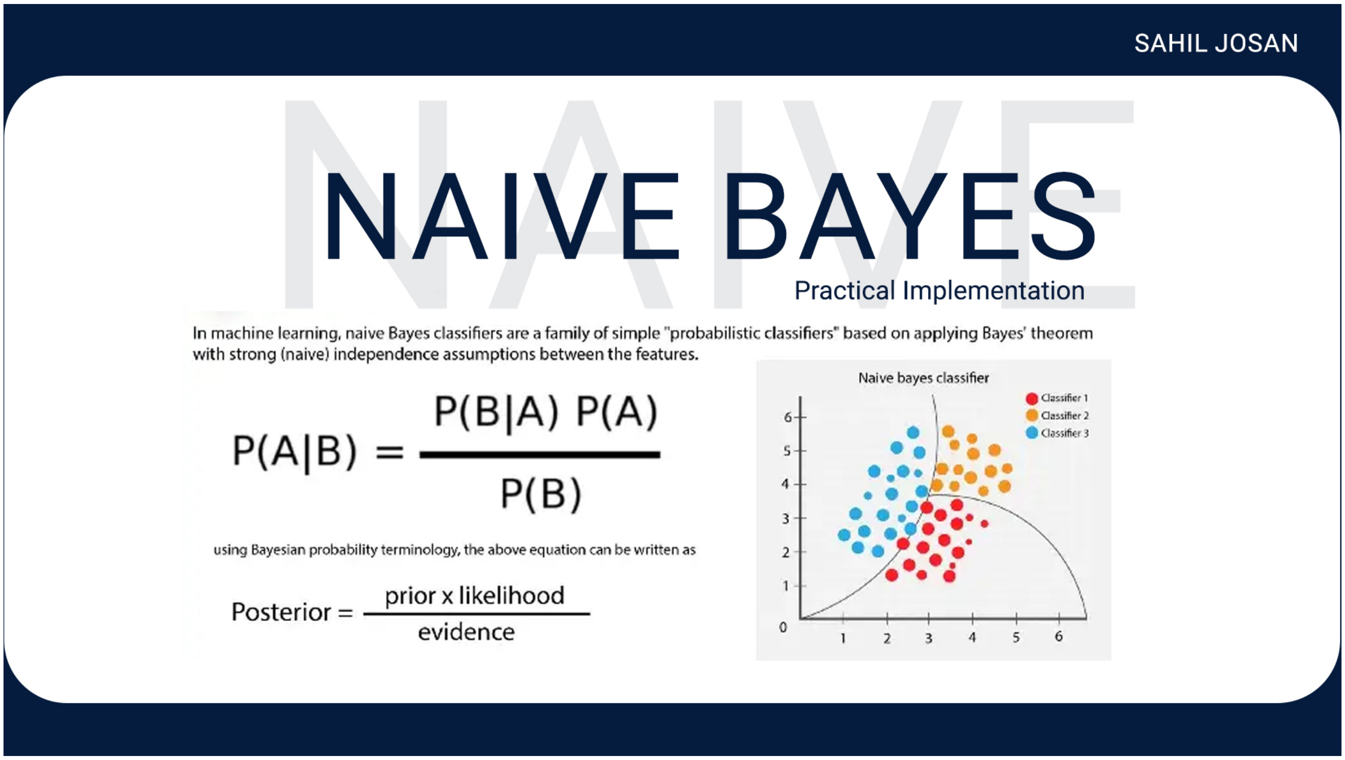
Naive Bayes Practical Implementation



SAHIL JOSAN

NAIVE BAYES

Practical Implementation

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

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

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$

In [1]:

```python
import pandas as pd
import numpy as np
```

In [2]:

```python
from sklearn import datasets
wine = datasets.load_wine()
```

In [3]:

```python
print(wine)
```

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
        1.185e+03],
       ...,
       [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
        5.600e+02]]), 'target': array([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, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2]), 'frame': None, 'target_names': array(['class_0', 'class_1', 'class_2'], d
type='<U7'), 'DESCR': '.. _wine_dataset:\n\nWine recognition dataset\n------------------
------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 178\n    :Number of
Attributes: 13 numeric, predictive attributes and the class\n    :Attribute Informatio
n:\n \t\t- Alcohol\n \t\t- Malic acid\n \t\t- Ash\n\t\t- Alcalinity of ash  \n \t\t- Mag
nesium\n\t\t- Total phenols\n \t\t- Flavanoids\n \t\t- Nonflavanoid phenols\n \t\t- Proa
nthocyanins\n\t\t- Color intensity\n \t\t- Hue\n \t\t- OD280/OD315 of diluted wines\n \t
\t- Proline\n\n    - class:\n            - class_0\n            - class_1\n            -
class_2\n\t\t\n    :Summary Statistics:\n    \n    ============================= ==== ==
=== ======= =====\n                                Min   Max   Mean     SD\n    =====
======================== ==== ===== ======= =====\n    Alcohol:                     11.
0  14.8    13.0   0.8\n    Malic Acid:                 0.74  5.80    2.34  1.12\n    A
sh:                         1.36  3.23    2.36  0.27\n    Alcalinity of Ash:
10.6  30.0    19.5   3.3\n    Magnesium:                 70.0 162.0    99.7  14.3\n
  Total Phenols:                0.98  3.88    2.29  0.63\n    Flavanoids:
    0.34  5.08    2.03  1.00\n    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
\n    Proanthocyanins:              0.41  3.58    1.59  0.57\n    Colour Intensity:
         1.3  13.0     5.1   2.3\n    Hue:                          0.48  1.71    0.96
0.23\n    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71\n    Proline:
             278  1680     746   315\n    ============================= ==== ===== =====
== =====\n\n    :Missing Attribute Values: None\n    :Class Distribution: class_0 (59),
class_1 (71), class_2 (48)\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MAR
SHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThis is a copy of UCI ML Wine recog
nition datasets.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.da
ta\n\nThe data is the results of a chemical analysis of wines grown in the same\nregion
in Italy by three different cultivators. There are thirteen different\nmeasurements take
n for different constituents found in the three types of\nwine.\n\nOriginal Owners: \n\n
Forina, M. et al, PARVUS - \nAn Extendible Package for Data Exploration, Classification
and Correlation. \nInstitute of Pharmaceutical and Food Analysis and Technologies,\nVia
Brigata Salerno, 16147 Genoa, Italy.\n\nCitation:\n\nLichman, M. (2013). UCI Machine Lea
rning Repository\n[https://archive.ics.uci.edu/ml]. Irvine, CA: University of Californi
a,\nSchool of Information and Computer Science. \n\n.. topic:: References\n\n  (1) S. Ae
berhard, D. Coomans and O. de Vel, \n  Comparison of Classifiers in High Dimensional Set
tings, \n  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of  \n  Mat
hematics and Statistics, James Cook University of North Queensland. \n  (Also submitted
to Technometrics). \n\n  The data was used with many others for comparing various \n  cl
assifiers. The classes are separable, though only RDA \n  has achieved 100% correct clas
sification. \n  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data)) \n
(All results using the leave-one-out technique) \n\n  (2) S. Aeberhard, D. Coomans and
O. de Vel, \n  "THE CLASSIFICATION PERFORMANCE OF RDA" \n  Tech. Rep. no. 92-01, (1992),
Dept. of Computer Science and Dept. of \n  Mathematics and Statistics, James Cook Univer
sity of North Queensland. \n  (Also submitted to Journal of Chemometrics).\n', 'feature_
names': ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenol
s', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
'od280/od315_of_diluted_wines', 'proline']}
```

Extract features from the above wine array

In [4]:

```
print(wine.feature_names)
```

```
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'fla
vanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od
315_of_diluted_wines', 'proline']
```

In [5]:

```
print(wine.target_names)
```

```
['class_0' 'class_1' 'class_2']
```

In [6]:

```
X = pd.DataFrame(wine['data'])
X.head()
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0

4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

In [7]:

```
y = (wine['target'])
y
```

Out[7]:

```
array([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, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2])
```

In [8]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y, test_size=0.30,random_state=100)
```

In [9]:

```
x_train.shape,y_train.shape
```

Out[9]:

```
((124, 13), (124,))
```

Import GaussianNB

In [10]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train,y_train)
y_pred = gnb.predict(x_test)
print(y_pred)
```

```
[1 2 0 1 2 2 1 1 1 1 2 1 2 2 2 0 2 0 1 0 2 0 1 1 0 0 1 1 1 2 2 1 0 1 2 2 1
 1 2 2 0 2 2 2 0 2 2 2 0 0 0 1 0 1]
```

In [11]:

```
from sklearn import metrics
metrics.accuracy_score(y_test,y_pred)
```

Out[11]:

```
1.0
```

In [12]:

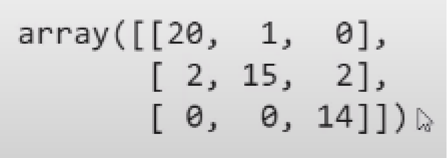
```
from sklearn.metrics import confusion_matrix
cm = np.array(confusion_matrix(y_test,y_pred))

cm
```

Out[12]:

```
array([[14,  0,  0],
       [ 0, 19,  0],
       [ 0,  0, 21]], dtype=int64)
```

- Here we can see our model has predicted all correct values, because in confusion matrix all diagonal values are correct values and any values other then diagonal values are incorrect. Like in our case all non-diagonal values are zero ..

  Another Example



```
array([[20,  1,  0],
       [ 2, 15,  2],
       [ 0,  0, 14]])
```

  Here in this example the model has predicted 5 wrong values. Because the diagonal values are for correct assumptons and values other then diagonal are wrong values

In this example 2+2+1 = 5, So 5 are wrong prediction and rest are current predictions

Naive Bayes' Classification on text data

Spam-Classification

In [13]:
```
Dataset = "https://raw.githubusercontent.com/sunnysavita10/Naive-Bayes/main/SpamClassifi
```

In [14]:
```
data = pd.read_csv(Dataset,sep="\t",header = None,names=['label',"messages"])
data
```

Out[14]:

	label	messages
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [15]:
```
data['messages'][0]
```

Out[15]:
```
'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cin
e there got amore wat...'
```

In [16]:
```
# NLTK: Natural Language Toolkit
import nltk
import re
```

Regular Expression `re:` it is used for finding out pattern in our text dataset

In [17]:
```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jayes\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[17]:
```
True
```

`Stop words` are a set of commonly used words in a language. Examples of stop words in English are `“a”`, `“the”`, `“is”`, `“are”` and etc. Stop words are commonly used in Text Mining and Natural Language Processing (NLP) to eliminate words that are so commonly used that they carry very little useful information.

In [18]:

```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

In [19]:

```
# For stemming
ps = PorterStemmer()
```

In [20]:

```
data['messages'][0]
```

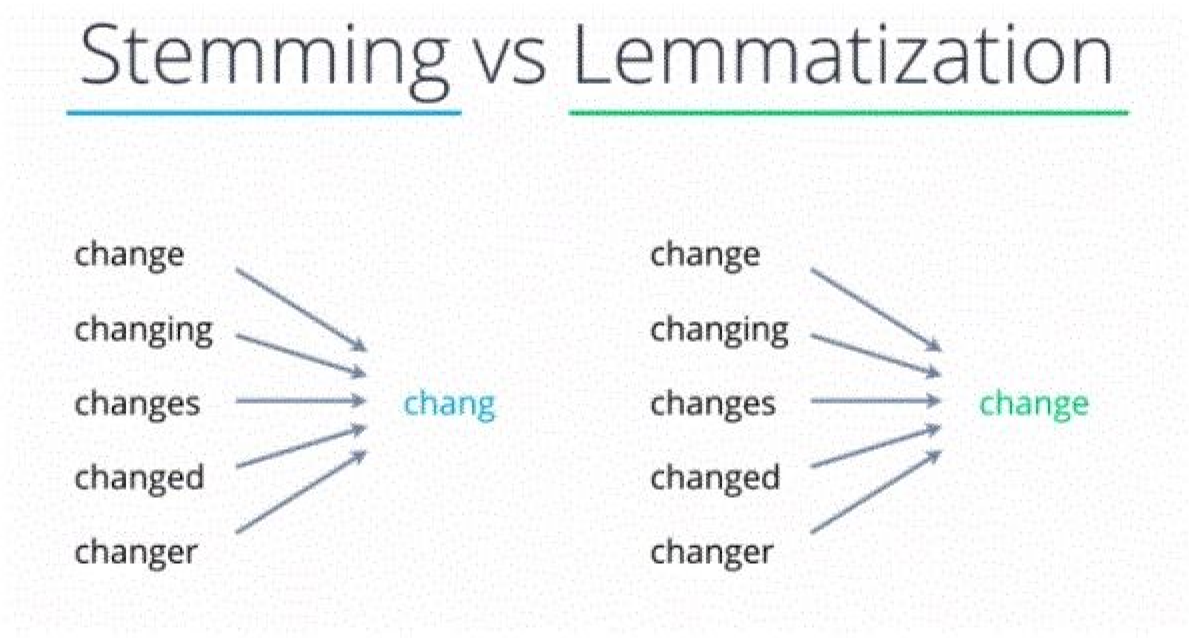
Out[20]:

```
'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cin
e there got amore wat...'
```

Clean the data and then do vectorization

Cleaning the data includes

- removing "a","e" or any this type of single character
- make all character in the dataset `lower`
- do stemming of the data



Lemitization takes more time compare to stemming

In [21]:

```
corpus = []
for i in range(0,len(data)):
    review = re.sub('[^a-z,A-Z]',' ',data['messages'][i])
    review = review.lower()
    review = review.split()

    review = [ps.stem(word) for word in review if not word in stopwords.words('english')
    review = ' '.join(review)
    corpus.append(review)
```

Now we have appended everything into our corpus

In [22]:

```
len(corpus)
```

Out[22]:

```
5572
```

In [23]:

```
corpus[:20]
```

Out[23]:

```
['go jurong point, crazi avail bugi n great world la e buffet cine got amor wat',
 'ok lar joke wif u oni',
 'free entri wkli comp win fa cup final tkt st may text fa receiv entri question std txt
rate c appli',
 'u dun say earli hor u c alreadi say',
 'nah think goe usf, live around though',
 'freemsg hey darl week word back like fun still tb ok xxx std chg send, rcv',
 'even brother like speak treat like aid patent',
 'per request mell mell oru minnaminungint nurungu vettam set callertun caller press cop
i friend callertun',
 'winner valu network custom select receivea prize reward claim call claim code kl valid
hour',
 'mobil month u r entitl updat latest colour mobil camera free call mobil updat co fre
e',
 'gonna home soon want talk stuff anymor tonight, k cri enough today',
 'six chanc win cash , pound txt csh send cost p day, days, tsandc appli repli hl info',
 'urgent week free membership , prize jackpot txt word claim c www dbuk net lccltd pobox
ldnw rw',
 'search right word thank breather promis wont take help grant fulfil promis wonder bles
s time',
 'date sunday',
 'xxxmobilemovieclub use credit, click wap link next txt messag click http wap xxxmobile
movieclub com n qjkgighjjgcbl',
 'oh k watch',
 'eh u rememb spell name ye v naughti make v wet',
 'fine way u feel way gota b',
 'england v macedonia dont miss goal team news txt ur nation team eg england tri wales,
scotland txt poboxox w wq']
```

Convert this data into vector

We go in sklearn.feature_extraction

Here we can use any vectoriser to convert our data into vector

In [24]:

```python
from sklearn.feature_extraction.text import CountVectorizer
```

In [25]:

```python
cv = CountVectorizer()
```

In [26]:

```python
X = cv.fit_transform(corpus).toarray()
X
```

Out[26]:

```
array([[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]], dtype=int64)
```

Now our data is converted into vectors

In [27]:

```python
X.shape
# 5572 rows
# 6531 columns
```

Out[27]:

```
(5572, 6531)
```

WE CAN REDUCE THE NUMBER OF FEATURES BY USING max_features PARAMETER

In [28]:

```python
cv = CountVectorizer(max_features = 2500)
X1 = cv.fit_transform(corpus).toarray()
X1
```

```
Out[28]: array([[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]], dtype=int64)
```

```
In [29]: X1.shape
```

```
Out[29]: (5572, 2500)
```

Our target column is:

```
In [30]: data['label']
```

```
Out[30]: 0         ham
         1         ham
         2        spam
         3         ham
         4         ham
                  ...
         5567     spam
         5568      ham
         5569      ham
         5570      ham
         5571      ham
         Name: label, Length: 5572, dtype: object
```

```
In [31]: pd.get_dummies(data['label'])
```

Out[31]:

	ham	spam
0	1	0
1	1	0
2	0	1
3	1	0
4	1	0
...	...	...
5567	0	1
5568	1	0
5569	1	0
5570	1	0
5571	1	0

5572 rows × 2 columns

Code To get only spam feature

```
In [32]: y = pd.get_dummies(data['label'], drop_first=True)
         y
```

Out[32]:

	spam
0	0

1	0
2	1
3	0
4	0
...	...
5567	1
5568	0
5569	0
5570	0
5571	0

5572 rows × 1 columns

Train Test Split

In [33]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=10)
```

In [34]:
```
# Model training
from sklearn.naive_bayes import GaussianNB
model  = GaussianNB()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print(y_pred)
```

```
C:\Users\jayes\anaconda3\envs\env\lib\site-packages\sklearn\utils\validation.py:1143: Da
taConversionWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
[0 0 0 ... 0 0 1]
```

In [35]:
```
# Check Accuracy
from sklearn import metrics
print("Accuracy by GaussianNB: ",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy by GaussianNB:  0.8743718592964824
```

Check accuracy with multinomialNB

In [36]:
```
from sklearn.naive_bayes import MultinomialNB
model2 = MultinomialNB()
model2.fit(x_train,y_train)
y_pred2 = model2.predict(x_test)
print("Accuracy Score by MulinomialNB: ",metrics.accuracy_score(y_test,y_pred2))
```

```
C:\Users\jayes\anaconda3\envs\env\lib\site-packages\sklearn\utils\validation.py:1143: Da
taConversionWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy Score by MulinomialNB:  0.9691313711414213
```

In [37]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download Naive_Bayes_practical_implement
```

```
[NbConvertApp] Converting notebook Naive_Bayes_practical_implementation.ipynb to webpdf
[NbConvertApp] Building PDF
```

```
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 608874 bytes to Naive_Bayes_practical_implementation.pdf
```

In [ ]:

```
```