# Naive Baye's Algorithm (classification)

1. Probability
2. Baye's Theorem

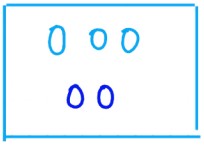
## Independent Events

Rolling a Dice $\{1, 2, 3, 4, 5, 6\}$

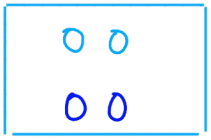
$Pr(1) = \frac{1}{6}$ $Pr(2) = \frac{1}{6}$ $Pr(3) = \frac{1}{6}$.

## Dependent Events

① What is the probability of removing a orange marble and then a yellow marble




↳ $P(O) = \frac{3}{5}$ → 1st Event → Orange marble has been Removed




→ $P(Y/O) = \frac{2}{4} = \frac{1}{2}$ ⇒ 2nd Event ⇒ Removed the Yellow Marble

$P(O \text{ and } Y) = P(O) * P(Y/O)$ → Conditional Probability

(I st, II nd)

$= 3/5 * \frac{1}{2} = \boxed{\frac{3}{10}}$

$$P(A \text{ and } B) = P(A) * P(B/A)$$

# Bayes Theorem

$$P(A \text{ and } B) = P(B \text{ and } A)$$

$$P(A) * P(B/A) = P(B) * P(A/B)$$

$$P(A/B) = \frac{P(A) * P(B/A)}{P(B)} \Rightarrow \text{Bayes Theorem.}$$

$P(A/B)$ : Probability of Event A given B has occured

$P(A)$ : Probability of Event A

$P(B)$ : Probability of Event B

$P(B/A)$ : Probability of Event B given A has occured.

$$P(A/B) = \frac{P(A) * P(B/A)}{P(B)} \Rightarrow \text{Bayes Theorem.}$$

I/P features ↓ Dependent

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| – | – | – | Yes |
| – | – | – | No |
| – | – | – | Yes |
| – | – | – | No |

$$P(y/(x_1,x_2,x_3)) = \frac{P(y) * P(x_1,x_2,x_3)/y}{P(x_1,x_2,x_3)}$$

$$P(y/(x_1,x_2,x_3)) = \frac{P(y) * P(x_1,x_2,x_3)/y}{P(x_1,x_2,x_3)}$$

$$= \frac{P(y) * P(x_1/y) * P(x_2/y) * P(x_3/y)}{P(x_1) * P(x_2) * P(x_3)}$$

I/P features ↓ Dependent

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| – | – | – | Yes |
| – | – | – | No |
| – | – | – | Yes |
| – | – | – | No |

[0.60, 0.40]
⇓
Yes

New test data

$$Pr(Yes/(x_1,x_2,x_3)) = \frac{P(Yes) * P(x_1/Yes) * P(x_2/Yes) * P(x_3/Yes)}{P(x_1) * Pr(x_2) * Pr(x_3)} \Rightarrow \text{Constant} = 0.60$$

⇓
Yes

$$Pr(No/(x_1,x_2,x_3)) = \frac{P(No) * Pr(x_1/No) * Pr(x_2/No) * Pr(x_3/No)}{P(x_1) * Pr(x_2) * Pr(x_3)} \Rightarrow \text{constant} = 0.40$$

# Lets Solve this Problem

| Day | Outlook | Temperature | Humidity | Wind | Play Tennis |
|---|---|---|---|---|---|
| 1 | Sunny | Hot | High | Weak | No |
| 2 | Sunny | Hot | High | Strong | No |
| 3 | Overcast | Hot | High | Weak | Yes |
| 4 | Rain | Mild | High | Weak | Yes |
| 5 | Rain | Cool | Normal | Weak | Yes |
| 6 | Rain | Cool | Normal | Strong | No |
| 7 | Overcast | Cool | Normal | Strong | Yes |
| 8 | Sunny | Mild | High | Weak | No |
| 9 | Sunny | Cool | Normal | Weak | Yes |
| 10 | Rain | Mild | Normal | Weak | Yes |
| 11 | Sunny | Mild | Normal | Strong | Yes |
| 12 | Overcast | Mild | High | Strong | Yes |
| 13 | Overcast | Hot | Normal | Weak | Yes |
| 14 | Rain | Mild | High | Strong | No |

## Outlook

| | Yes | No | P(E/Yes) | Pr(E/No) |
|---|---|---|---|---|
| Sunny | 2 | 3 | 2/9 | 3/5 |
| Overcast | 4 | 0 | 4/9 | 0/5 |
| Rain | 3 | 2 | 3/9 | 2/5 |

→ Test (Sunny, Hot) → O/P

## Temperature

| | Yes | No | P(E/Yes) | P(E/No) |
|---|---|---|---|---|
| Hot | 2 | 2 | 2/9 | 2/5 |
| Mild | 4 | 2 | 4/9 | 4/5 |
| Cool | 3 | 1 | 3/9 | 3/5 |

## PLAY (Y/N)

| | | P(Yes) | P(No) |
|---|---|---|---|
| Yes | 9 | 9/14 | 5/14 |
| No | 5 | | |

$$P(\text{Yes}/\text{Sunny}, \text{Hot}) = \frac{P(\text{Yes}) * Pr(\text{Sunny}/\text{Yes}) * Pr(\text{Hot}/\text{Yes})}{Pr(\text{Sunny}) * Pr(\text{Hot})}$$

$$= 9/14 * \frac{2}{9} * \frac{2}{9}$$

$$= \frac{2}{63} = 0.031$$

$$P(\text{No} \mid (\text{Sunny}, \text{Hot})) = P(\text{No}) * Pr(\text{Sunny}/\text{No}) * Pr(\text{Hot}/\text{No})$$

$$= 5/14 * 3/5 * 2/5$$

$$= \frac{3}{35} = 0.085$$

$$Pr(\text{Yes} / (\text{Sunny}, \text{Hot})) = \frac{0.031}{(0.031 + 0.085)} = 0.27 = 27\%$$

$$Pr(No \mid Sunny, hot) = \frac{0.085}{(0.031 + 0.085)} = 0.73 = 73\%$$

New Test =>

| Outlook | Temperature | O/P |
|---|---|---|
| Sunny | Hot | |

73% => They will not play Tennis

27% => They will play Tennis.

⇓

0 => Person is not going to playing Tennis

# Variants of Naive Baye's

1. Bernoulli Naive Bayes
2. Multinomial Naive Bayes
3. Gaussian Naive Bayes

## 1. Bernoulli Naive Bayes

Whenever your features are following a Bernoulli Distribution, then we need to use Bernoulli Naive Baye Algorithm.

Dataset

Bernoulli → 0, 1

| f1 | f2 | f3 | O/P |
|---|---|---|---|
| Yes | Pass | Male | Yes |
| Yes | Fail | Female | No |
| No | Pass | Male | Yes |
| Yes | Fail | Female | No |

## 2. Multinomial Naive Bayes ⇒ I/p = Text

Datasit : Spam classification

O/P

I/P → Email Body — Spam/Not Spam

feature

You have Million $$ lottery — Spam

KRISH YOU HAVE DONE GOOD JOB — HAM

↓

Numerical Values ⇒ Natural Language Procming

↓

Vectors

① BOW

② Tf-Idf

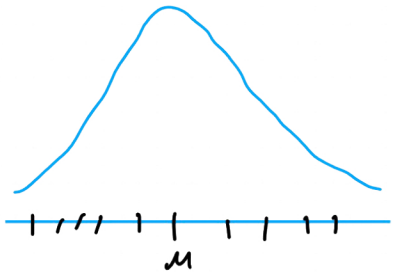
③ Word2vec

## ③ Gaussian Naive Bayes

If the features are following Gaussian Dismbution, then we use Gaussian Naive Bayes

DATASET → CONTINUOUS

IRIS Dataset




| Age | Height | Weight | Yes/No |
|---|---|---|---|
| 25 | 170 | 78 | |
| 38 | 160 | 75 | |
| 22 | 150 | 60 | |
| 24 | 170 | 35 | |

# ML_Naive_Bayes

## 1 Naive Baye's Machine Learning Algorithms Implementation

[3]:
```
from sklearn.datasets import load_iris
```

[4]:
```
from sklearn.model_selection import train_test_split
```

[5]:
```
X,y = load_iris(return_X_y=True)
## 'return_X_y' in sklearn.dataset return independent and dependent
### variables
## we can also first initiallize and store it seperately but
### this method is fastest and with less code
```

[6]:
```
dataset = load_iris()
```

[72]:
```
dataset ## like this we can store X and y independent and target data
## but for now we are using 'return_X_y=True'
```

[8]:
```
X,y = load_iris(return_X_y=True)

y ## 0, 1 and 2 means multi-class classification (3 categories)
```

[8]:
```
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, 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, 2, 2])
```

[9]:
```
## train test split
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.3,
                                                    random_state=0)
```

- in X_train we can see all the distributions are numerical values (continuous value)
- for continuous values we use Gaussian Naive Baye's

[10]:
```
from sklearn.naive_bayes import GaussianNB
```

[11]:
```
gnb = GaussianNB()
```

[12]:
```
gnb.fit(X_train,y_train)
```

[12]:
```
GaussianNB()
```

[13]:
```
y_pred = gnb.predict(X_test)
```

[14]:
```
## accuracy
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
```

[15]:
```
print(confusion_matrix(y_pred,y_test))
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
[[16  0  0]
 [ 0 18  0]
 [ 0  0 11]]
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        11

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

- we got perfect accuracy '1' because iris dataset is very small,
- if our dataset had bernoulli distribution we would have used Bernoulli Naive Bayes
- for text data or words/sentence we use Multinomial Naive Bayes or

[ ]:
```

```

### 1.0.1 trying with different Dataset

[42]:
```
import seaborn as sns
```

[43]:
```
df = sns.load_dataset('tips')
```

[44]:
```
##we'll predict time
df.head()
```

[44]:

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| 0 | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| 1 | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| 2 | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| 3 | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| 4 | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

[45]:
```
## converting categorical data to numerical using One Hot Encoding
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
```

[46]:
```
encoder.fit_transform(df[['sex','smoker','day','time']]).toarray()[0:5] ##␣
 ↪print top 5 encoded row
```

[46]:
```
array([[1., 0., 1., 0., 0., 0., 1., 0., 1., 0.],
       [0., 1., 1., 0., 0., 0., 1., 0., 1., 0.],
       [0., 1., 1., 0., 0., 0., 1., 0., 1., 0.],
       [0., 1., 1., 0., 0., 0., 1., 0., 1., 0.],
       [1., 0., 1., 0., 0., 0., 1., 0., 1., 0.]])
```

[47]:
```
encoded = encoder.fit_transform(df[['sex','smoker','day','time']]).toarray()
```

[48]:
```
import pandas as pd
encoder_df = pd.DataFrame(encoded, columns=encoder.get_feature_names_out())
```

[49]:
```
encoder_df.head()
```

[49]:

| | sex_Female | sex_Male | smoker_No | smoker_Yes | day_Fri | day_Sat | day_Sun | day_Thur | time_Dinner | time_Lunch |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 1 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 2 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 3 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 4 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |

[50]:
```
pd.concat([df,encoder_df],axis=1)[0:5]## print top 5 row
```

[50]:

| | total_bill | tip | sex | smoker | day | time | size | sex_Female | sex_Male |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 | 1.0 | 0.0 |
| 1 | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 | 0.0 | 1.0 |
| 2 | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 | 0.0 | 1.0 |

```
3       23.68  3.31    Male     No  Sun  Dinner     2         0.0       1.0
4       24.59  3.61  Female     No  Sun  Dinner     4         1.0       0.0

   smoker_No  smoker_Yes  day_Fri  day_Sat  day_Sun  day_Thur  time_Dinner  \
0        1.0         0.0      0.0      0.0      1.0       0.0          1.0
1        1.0         0.0      0.0      0.0      1.0       0.0          1.0
2        1.0         0.0      0.0      0.0      1.0       0.0          1.0
3        1.0         0.0      0.0      0.0      1.0       0.0          1.0
4        1.0         0.0      0.0      0.0      1.0       0.0          1.0

   time_Lunch
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
```

[51]:
```
## or we can use
df.join(encoder_df)[0:5]## print top 5 row
```

[51]:
```
   total_bill   tip     sex smoker  day    time  size  sex_Female  sex_Male  \
0       16.99  1.01  Female     No  Sun  Dinner     2         1.0       0.0
1       10.34  1.66    Male     No  Sun  Dinner     3         0.0       1.0
2       21.01  3.50    Male     No  Sun  Dinner     3         0.0       1.0
3       23.68  3.31    Male     No  Sun  Dinner     2         0.0       1.0
4       24.59  3.61  Female     No  Sun  Dinner     4         1.0       0.0

   smoker_No  smoker_Yes  day_Fri  day_Sat  day_Sun  day_Thur  time_Dinner  \
0        1.0         0.0      0.0      0.0      1.0       0.0          1.0
1        1.0         0.0      0.0      0.0      1.0       0.0          1.0
2        1.0         0.0      0.0      0.0      1.0       0.0          1.0
3        1.0         0.0      0.0      0.0      1.0       0.0          1.0
4        1.0         0.0      0.0      0.0      1.0       0.0          1.0

   time_Lunch
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
```

[52]:
```
df_new = pd.concat([df,encoder_df],axis=1)
```

[53]:
```
df_new.columns
```

[53]:
```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size',
       'sex_Female', 'sex_Male', 'smoker_No', 'smoker_Yes', 'day_Fri',
```

```
       'day_Sat', 'day_Sun', 'day_Thur', 'time_Dinner', 'time_Lunch'],
      dtype='object')
```

[54]:
```
df_new.drop(['sex','smoker','day','time'],axis=1)[0:5]## print top 5 row
```

[54]:

| | total_bill | tip | size | sex_Female | sex_Male | smoker_No | smoker_Yes | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 16.99 | 1.01 | 2 | 1.0 | 0.0 | 1.0 | 0.0 | |
| 1 | 10.34 | 1.66 | 3 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 2 | 21.01 | 3.50 | 3 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 3 | 23.68 | 3.31 | 2 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 4 | 24.59 | 3.61 | 4 | 1.0 | 0.0 | 1.0 | 0.0 | |

| | day_Fri | day_Sat | day_Sun | day_Thur | time_Dinner | time_Lunch |
|---|---|---|---|---|---|---|
| 0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 1 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 2 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 3 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| 4 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |

[55]:
```
## to predict time we have to make time label encoded or categories encoded␣
 ↪because it's our dependent/target variable
df_new.drop(['sex','smoker','day','time','time_Dinner','time_Lunch'],axis=1)[0:
 ↪5]## print top 5 row
```

[55]:

| | total_bill | tip | size | sex_Female | sex_Male | smoker_No | smoker_Yes | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 16.99 | 1.01 | 2 | 1.0 | 0.0 | 1.0 | 0.0 | |
| 1 | 10.34 | 1.66 | 3 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 2 | 21.01 | 3.50 | 3 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 3 | 23.68 | 3.31 | 2 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 4 | 24.59 | 3.61 | 4 | 1.0 | 0.0 | 1.0 | 0.0 | |

| | day_Fri | day_Sat | day_Sun | day_Thur |
|---|---|---|---|---|
| 0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 1 | 0.0 | 0.0 | 1.0 | 0.0 |
| 2 | 0.0 | 0.0 | 1.0 | 0.0 |
| 3 | 0.0 | 0.0 | 1.0 | 0.0 |
| 4 | 0.0 | 0.0 | 1.0 | 0.0 |

[56]:
```
X = df_new.drop(['sex','smoker','day','time','time_Dinner','time_Lunch'],axis=1)
```

[57]:
```
X.head()
```

[57]:

| | total_bill | tip | size | sex_Female | sex_Male | smoker_No | smoker_Yes | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 16.99 | 1.01 | 2 | 1.0 | 0.0 | 1.0 | 0.0 | |
| 1 | 10.34 | 1.66 | 3 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 2 | 21.01 | 3.50 | 3 | 0.0 | 1.0 | 1.0 | 0.0 | |
| 3 | 23.68 | 3.31 | 2 | 0.0 | 1.0 | 1.0 | 0.0 | |

```
4       24.59  3.61     4          1.0        0.0         1.0          0.0

   day_Fri  day_Sat  day_Sun  day_Thur
0      0.0      0.0      1.0       0.0
1      0.0      0.0      1.0       0.0
2      0.0      0.0      1.0       0.0
3      0.0      0.0      1.0       0.0
4      0.0      0.0      1.0       0.0
```

[58]:
```
df.time[0:5]## print top 5 row
```

[58]:
```
0    Dinner
1    Dinner
2    Dinner
3    Dinner
4    Dinner
Name: time, dtype: category
Categories (2, object): ['Lunch', 'Dinner']
```

[59]:
```
df.time.cat.codes[76:91] ## checking dinner = 1 and lunch =0
```

[59]:
```
76    1
77    0
78    0
79    0
80    0
81    0
82    0
83    0
84    0
85    0
86    0
87    0
88    0
89    0
90    1
dtype: int8
```

[60]:
```
df[76:91]
```

[60]:
```
    total_bill   tip     sex smoker   day    time  size
76       17.92  3.08    Male    Yes   Sat  Dinner     2
77       27.20  4.00    Male     No  Thur   Lunch     4
78       22.76  3.00    Male     No  Thur   Lunch     2
79       17.29  2.71    Male     No  Thur   Lunch     2
80       19.44  3.00    Male    Yes  Thur   Lunch     2
81       16.66  3.40    Male     No  Thur   Lunch     2
```

```
82       10.07  1.83  Female     No  Thur   Lunch     1
83       32.68  5.00    Male    Yes  Thur   Lunch     2
84       15.98  2.03    Male     No  Thur   Lunch     2
85       34.83  5.17  Female     No  Thur   Lunch     4
86       13.03  2.00    Male     No  Thur   Lunch     2
87       18.28  4.00    Male     No  Thur   Lunch     2
88       24.71  5.85    Male     No  Thur   Lunch     2
89       21.16  3.00    Male     No  Thur   Lunch     2
90       28.97  3.00    Male    Yes   Fri  Dinner     2
```

[61]:
```
y = df.time.cat.codes ## convert it to categories code 0 and 1, we can also use␣
↪label encoding
```

[62]:
```
y[76:91]
```

[62]:
```
76    1
77    0
78    0
79    0
80    0
81    0
82    0
83    0
84    0
85    0
86    0
87    0
88    0
89    0
90    1
dtype: int8
```

[63]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25,␣
↪random_state=0)
```

[65]:
```
X_train.head()
```

[65]:
```
     total_bill   tip  size  sex_Female  sex_Male  smoker_No  smoker_Yes  \
171       15.81  3.16     2         0.0       1.0        0.0         1.0
89        21.16  3.00     2         0.0       1.0        1.0         0.0
157       25.00  3.75     4         1.0       0.0        1.0         0.0
219       30.14  3.09     4         1.0       0.0        0.0         1.0
234       15.53  3.00     2         0.0       1.0        0.0         1.0

     day_Fri  day_Sat  day_Sun  day_Thur
171      0.0      1.0      0.0       0.0
89       0.0      0.0      0.0       1.0
```

```
157      0.0      0.0      1.0      0.0
219      0.0      1.0      0.0      0.0
234      0.0      1.0      0.0      0.0
```

[66]:
```
y_train.head()
```

[66]:
```
171    1
89     0
157    1
219    1
234    1
dtype: int8
```

[67]:
```
from sklearn.naive_bayes import GaussianNB, BernoulliNB
```

[68]:
```
gnb = GaussianNB()
gnb.fit(X_train,y_train)
y_pred = gnb.predict(X_test)
```

[69]:
```
print(confusion_matrix(y_pred,y_test))
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
[[11  1]
 [ 5 44]]
0.9016393442622951
              precision    recall  f1-score   support

           0       0.69      0.92      0.79        12
           1       0.98      0.90      0.94        49

    accuracy                           0.90        61
   macro avg       0.83      0.91      0.86        61
weighted avg       0.92      0.90      0.91        61
```

[ ]:
```

```

[70]:
```
bnb = BernoulliNB()
bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
```

[71]:
```
print(confusion_matrix(y_pred,y_test))
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
[[11  1]
```

```
 [ 5 44]]
0.9016393442622951
              precision    recall  f1-score   support

           0       0.69      0.92      0.79        12
           1       0.98      0.90      0.94        49

    accuracy                           0.90        61
   macro avg       0.83      0.91      0.86        61
weighted avg       0.92      0.90      0.91        61
```

[ ]: