Naive Bayes Classifier

- is a supervised learning model
- uses Bayes' Theorem
- works on conditional probability / posterior probability
- conditional probability, P(A|B) = P(A ∪ B)/P(A)
- used to find probability of event A if probability of event B is already known
- e.g. supermarket case, if user buys bread and eggs, what is probability of user buying milk
- e.g. if a person is wearing a saree is a feature, and we know the probability of person wearing saree, what will be probability of person being female ?

```
import numpy as np
import pandas as pd
import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
import numpy as np
import pandas as pd
import sklearn
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='iris')
df.head()
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('Species', axis=1), df.Species, test_size=0.25)
```

▼ GaussianNB().fit(x_train, y_train)

```
mod1 = GaussianNB().fit(x_train, y_train)
```

▼ mod1.predict(x_test)

```
y_pred = mod1.predict(x_test)
```

▼ confusion_matrix(y_test, y_pred)

```
print(confusion_matrix(y_test, y_pred))
```

```
[[12  0  0]
 [ 0 11  3]
 [ 0  0 12]]
```

```
1
```

▼ KNN classifier

- K-Nearest Neighbour classifier

- selects `k` number of neighbours, and then selects the response with highest frequency as the final response
-

import KNeighborsClassifier

```
import numpy as np
import pandas as pd
import sklearn
from sklearn.neighbors import KNeighborsClassifier as knc
from sklearn.model_selection import train_test_split
```

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='iris')
df.head()
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('Species', axis=1), df.Species, test_size=0.25)
```

KNeighborsClassifier(n_neighbors=4).fit(x_train, y_train)

```
mod2 = knc(n_neighbors=4).fit(x_train, y_train)
```

model.predict(x_test)

```
y_pred2 = mod2.predict(x_test)
```

confusion_matrix(y_test, y_pred)

```
print(confusion_matrix(y_test, y_pred))
# confusion matrix for naive Bayes classifier
```

```
[[6 4 3]
 [2 4 4]
 [4 3 8]]
```

```
print(confusion_matrix(y_test, y_pred2))
# confusion matrix for KNN classifier
```

```
[[13  0  0]
 [ 0 10  0]
 [ 0  1 14]]
```

- ROC curve: receiver operating characteristic curve
- indicates preicision, recall value
- higher the area under the curve, higher the precision

Unsupervised learning Models

- there is no response in training data
- used to discover hidden patterns
- uses machine learning algorithms to analyze and cluster unlabeled datasets

Clustering Model

- we make atleast three clusters
- each cluster has one center point called centeroid
- one point in a specific cluster is closer to the centeroid of that cluster as compared to the centroid of other clusters in dataset
- majorly used in marketing industry to analyse

Partitional Clustering

K-Means

- is a clustering model, unsupervised learning model
-

import KMeans

```
import numpy as np
import pandas as pd
import sklearn
from sklearn.cluster import KMeans
```

```
from google.colab import files
uploaded=files.upload()
# clusters.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving clusters.xlsx to clusters.xlsx

```
df = pd.read_excel('clusters.xlsx')
df.head()
```

KMeans(n_clusters=3).fit(dataSet)

```
mod3 = KMeans(n_clusters=3).fit(df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
  warnings.warn(
```

model.cluster_centers_

```
mod3.cluster_centers_
# prints centroids from each of the clusters
```

```
array([[43.2, 16.7],
       [29.6, 66.8],
       [55.1, 46.1]])
```

mod.labels_

```
mod3.labels_
# prints the cluster to which each of the record is associated
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

model.predict(2-D List)

```
mod3.predict([[47, 14]])
# returns the array index of the cluster to whose centroid,
# this point is closest to
```

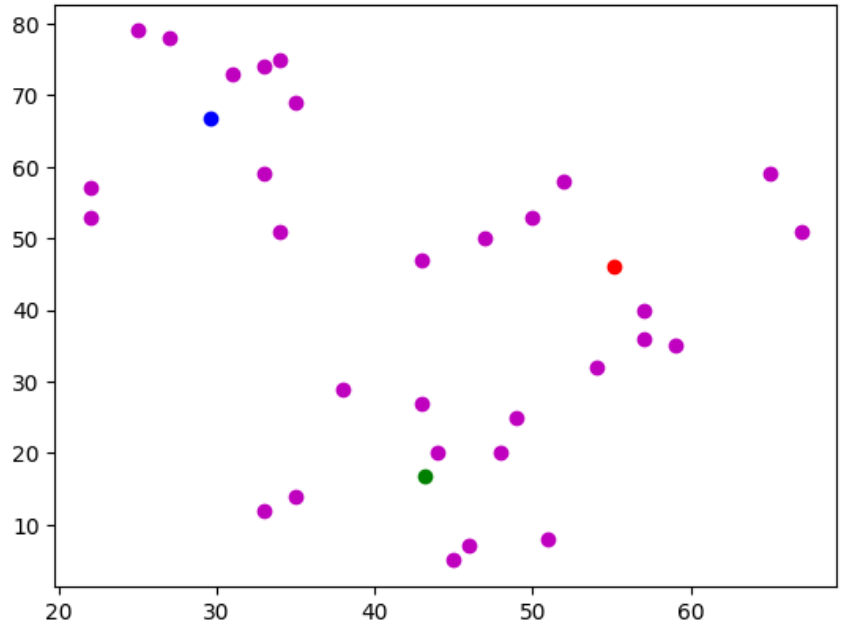
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but KMeans was fitted with feature names
  warnings.warn(
array([0], dtype=int32)
```

```
from matplotlib import pyplot as plt
```

```
plt.scatter(df.Var1, df.Var2, c='m') # all the points from dataset
plt.scatter(43.2, 16.7, c='g') # centroid1
```

```
plt.scatter(29.6, 66.8, c='b') # centroid2
plt.scatter(55.1, 46.1, c='r') # centroid2
```

```
<matplotlib.collections.PathCollection at 0x7f6b44965960>
```



Deciding on number of clusters

- maximum possible number of cluster is total number of records in dataset, so that each record/point is a unique cluster
- if number of clusters increase, the variation within the cluster will decrease

```
mod4 = KMeans(n_clusters=4).fit(df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
  warnings.warn(
```

```
mod6 = KMeans(n_clusters=6).fit(df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
  warnings.warn(
```

```
mod8 = KMeans(n_clusters=8).fit(df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
  warnings.warn(
```

```
mod8.cluster_centers_
```

```
array([[27.75      , 55.        ],
       [47.33333333,  6.66666667],
       [56.75      , 35.75      ],
       [30.83333333, 74.66666667],
       [34.        , 13.        ],
       [48.        , 52.        ],
       [66.        , 55.        ],
       [44.4       , 24.2       ]])
```

model.inertia_

- prints distance / gain / delta of points from their respective centroid, means how well a dataset was clustered by K-Means
- inertia / gain / delta should be lower value
- number of clusters should be low
- gain / delta comes down as you increase the number of cluster

```
mod3.inertia_
```

```
3649.5
```

```
mod6.inertia_
```

```
1389.2166666666667
```

```
mod8.inertia_
```

```
683.75
```

```
mod10 = KMeans(n_clusters=10).fit(df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
  warnings.warn(
```

```
mod10.inertia_
```

```
437.3333333333333
```

Hierarchical clustering

- output resembles or
- output might be dendrogram (tree-structure),
- finds out which items have similar attributes and creates clusters
- clusters at lowest level will have least differences, and as you go higher in level/altitude of levels, differences will increase
- top/highest level cluster will have all the items / items with maximum possible diffferences

import scipy.cluster.hierarchy

```
import scipy
import scipy.cluster.hierarchy as sch
```

```
from google.colab import files
uploaded=files.upload()
# CDAC_DataBook.xlsx
# to be used with google colab

# import os
# os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
# os.getcwd()
# to change current working directory to specified path
# to be used while running on local system
```

```
df = pd.read_excel('CDAC_DataBook.xlsx', sheet_name='mtcars')
df.head()
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

sch.dendrogram(sch.linkage(df))

```
sch.dendrogram(sch.linkage(df))
```

```
{'icoord': [[25.0, 25.0, 35.0, 35.0],
  [15.0, 15.0, 30.0, 30.0],
  [55.0, 55.0, 65.0, 65.0],
  [45.0, 45.0, 60.0, 60.0],
  [75.0, 75.0, 85.0, 85.0],
  [105.0, 105.0, 115.0, 115.0],
  [95.0, 95.0, 110.0, 110.0],
  [125.0, 125.0, 135.0, 135.0]
```

```
df.iloc[9]
```

```
mpg      19.20
cyl       6.00
disp    167.60
hp      123.00
drat      3.92
wt        3.44
qsec     18.30
vs        1.00
am        0.00
gear      4.00
carb      4.00
Name: 9, dtype: float64
```

```
[215.0, 215.0, 257.34375, 257.34375]
```

Time-Series Modelling

- special form of regression
- only one predictor
- response only depends on time
- variations over time can be analysed
- sequencing of data is extremely important in Time Series
  - order of records is not important in supervised or unsupervised modelling, but in time series modelling, ordering of records matter
  - sequencing of the records is extremely important, so data should be strictly as per time, otherwise results will change
- → Trend: slow variation over a longer period of time
  - e.g. sales of innova increased from its launch in 2005, and sales of qualis decreased from 2005
- → Seasonal variation: varaiation over smaller period of time
  - e.g. variation in car sales over different seasons
- → cyclical variation: when events occur after a fixed time period of years/centuries
  - e.g. 1919 spanish flu, 2019 COVID-19
- → irregular variations: random variations

15.29568648345016

preparing data for Time Series Modelling

- earlier we could remove null/duplicate records for supervised/unsupervised models, but if we remove such records, it'll impact the results, so we cannot remove such records

1. missing values need to be filled with some data
    - better to take data from previous year than copying data from last month
2. outliers need to be checked carefully, as outliers might contain some useful data -

- Error of the forecast = actual Value - Forecast Value

Types of errors

- MAD - Mean Absolute Deviation
- MSQ - Mean Square Error
- MAPE - Mean Absolute Percentage Error
- RMSE - Root Mean Square Error

```
1
```