Machine Learning (CS351) Project

Compare Various classifiers for water Quality Data

Submitted By

Gurubasavaraj BG (181CO120)

Ankit Raj (181CO106)

1. Introduction

Problem Statement

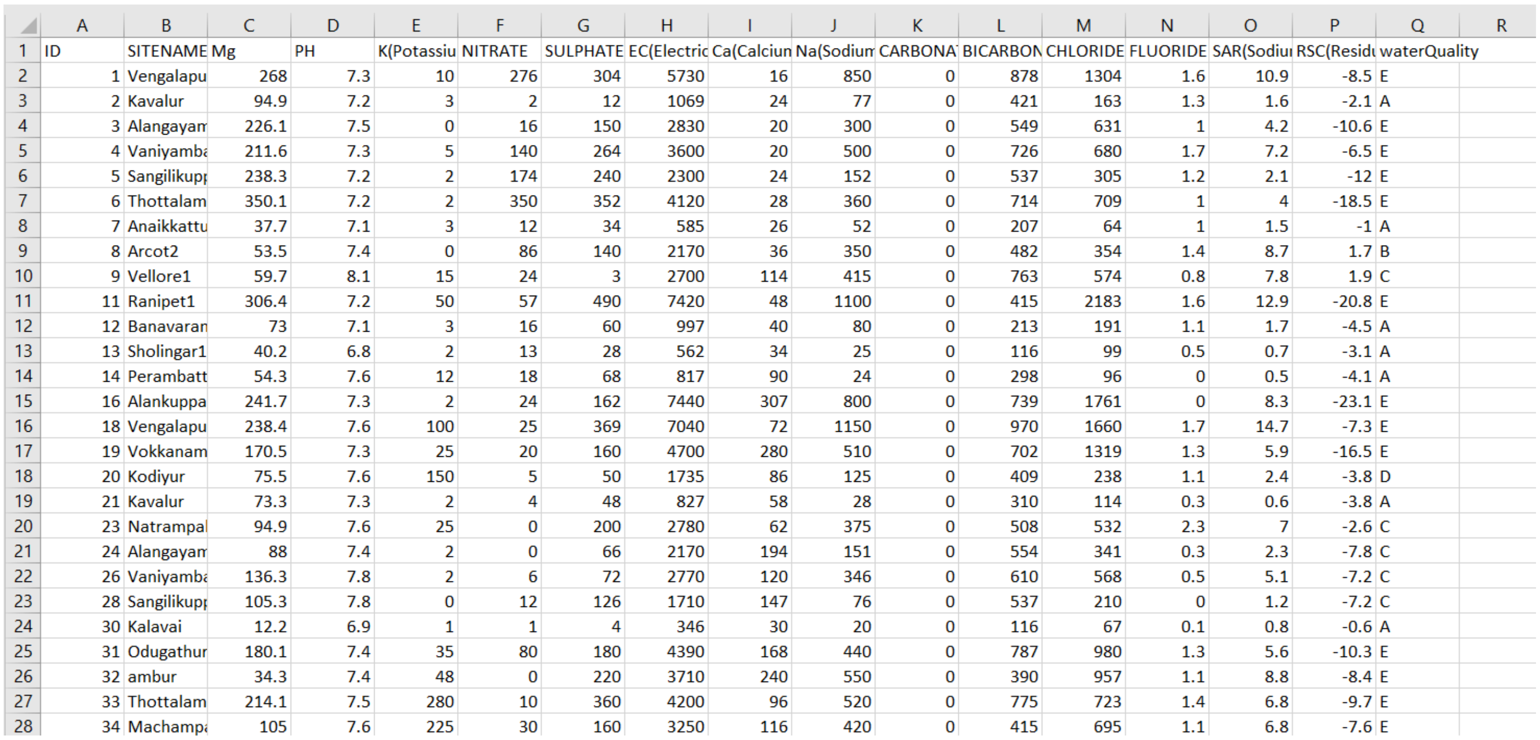
The problem statement is, we have given data of quality of water and factors responsible for that and we have to classify the data using classifier algorithms and then compare the data of result obtained in each algorithm.

Classifiers in Machine Learning: -
A classifier in machine learning is an algorithm that automatically orders or categorizes data into one or more of a set of “classes.”

Different types of classifiers:
1. Linear Regression
2. Logistic Regression
3. K Nearest Neighbor (KNN)
4. Decision tree
5. Naive Bayes
6. Support Vector Machine (SVM)

Data Description: -

Dataset consists of 16 independent references and one output reference. Ref1 and Ref2 doesn't have any impact on water quality and also since ref 'CARBOHYDRATE' has zero value for all it doesn't affect the quality. We have water quality data of 88 different places.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	ID	SITENAME	Mg	PH	K(Potassiu	NITRATE	SULPHATE	EC(Electric	Ca(Calciun	Na(Sodiun	CARBONA	BICARBON	CHLORIDE	FLUORIDE	SAR(Sodiu	RSC(Resid	waterQuality	
2	1	Vengalapu	268	7.3	10	276	304	5730	16	850	0	878	1304	1.6	10.9	-8.5	E	
3	2	Kavalur	94.9	7.2	3	2	12	1069	24	77	0	421	163	1.3	1.6	-2.1	A	
4	3	Alangayan	226.1	7.5	0	16	150	2830	20	300	0	549	631	1	4.2	-10.6	E	
5	4	Vaniyamba	211.6	7.3	5	140	264	3600	20	500	0	726	680	1.7	7.2	-6.5	E	
6	5	Sangilikupp	238.3	7.2	2	174	240	2300	24	152	0	537	305	1.2	2.1	-12	E	
7	6	Thottalam	350.1	7.2	2	350	352	4120	28	360	0	714	709	1	4	-18.5	E	
8	7	Anaikkattu	37.7	7.1	3	12	34	585	26	52	0	207	64	1	1.5	-1	A	
9	8	Arcot2	53.5	7.4	0	86	140	2170	36	350	0	482	354	1.4	8.7	1.7	B	
10	9	Vellore1	59.7	8.1	15	24	3	2700	114	415	0	763	574	0.8	7.8	1.9	C	
11	11	Ranipet1	306.4	7.2	50	57	490	7420	48	1100	0	415	2183	1.6	12.9	-20.8	E	
12	12	Banavaran	73	7.1	3	16	60	997	40	80	0	213	191	1.1	1.7	-4.5	A	
13	13	Sholingar1	40.2	6.8	2	13	28	562	34	25	0	116	99	0.5	0.7	-3.1	A	
14	14	Perambatt	54.3	7.6	12	18	68	817	90	24	0	298	96	0	0.5	-4.1	A	
15	16	Alankuppa	241.7	7.3	2	24	162	7440	307	800	0	739	1761	0	8.3	-23.1	E	
16	18	Vengalapu	238.4	7.6	100	25	369	7040	72	1150	0	970	1660	1.7	14.7	-7.3	E	
17	19	Vokkanam	170.5	7.3	25	20	160	4700	280	510	0	702	1319	1.3	5.9	-16.5	E	
18	20	Kodiyur	75.5	7.6	150	5	50	1735	86	125	0	409	238	1.1	2.4	-3.8	D	
19	21	Kavalur	73.3	7.3	2	4	48	827	58	28	0	310	114	0.3	0.6	-3.8	A	
20	23	Natrampal	94.9	7.6	25	0	200	2780	62	375	0	508	532	2.3	7	-2.6	C	
21	24	Alangayan	88	7.4	2	0	66	2170	194	151	0	554	341	0.3	2.3	-7.8	C	
22	26	Vaniyamba	136.3	7.8	2	6	72	2770	120	346	0	610	568	0.5	5.1	-7.2	C	
23	28	Sangilikupp	105.3	7.8	0	12	126	1710	147	76	0	537	210	0	1.2	-7.2	C	
24	30	Kalavai	12.2	6.9	1	1	4	346	30	20	0	116	67	0.1	0.8	-0.6	A	
25	31	Odugathur	180.1	7.4	35	80	180	4390	168	440	0	787	980	1.3	5.6	-10.3	E	
26	32	ambur	34.3	7.4	48	0	220	3710	240	550	0	390	957	1.1	8.8	-8.4	E	
27	33	Thottalam	214.1	7.5	280	10	360	4200	96	520	0	775	723	1.4	6.8	-9.7	E	
28	34	Machampa	105	7.6	225	30	160	3250	116	420	0	415	695	1.1	6.8	-7.6	E	

2. Linear Regression

Linear regression makes predictions for continuous/real or numeric variables.
Linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.
The linear regression model provides a sloped straight line representing the relationship between the variables.
Multiple Linear regression: If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.
Linear Regression Line: A linear line showing the relationship between the dependent and independent variables is called a regression line.
When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.
R-squared method:
It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
It can be calculated from the below formula:

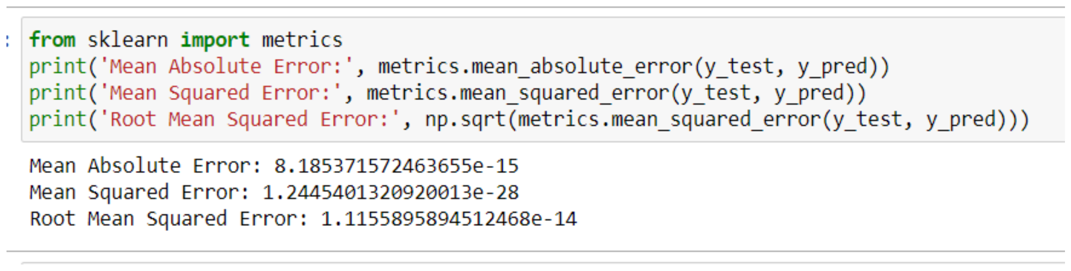
$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Linear Regression Implementation using Python

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings


df=pd.read_csv("dataset.csv")
df.head()

#dataset description
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
df.info()

df['waterQuality'] = df['waterQuality'].replace(['A'],'1')
df['waterQuality'] = df['waterQuality'].replace(['B'],'2')
df['waterQuality'] = df['waterQuality'].replace(['C'],'3')
df['waterQuality'] = df['waterQuality'].replace(['D'],'4')
df['waterQuality'] = df['waterQuality'].replace(['E'],'5')
df.head()

y = df['waterQuality'].astype('int32')
x = df.drop(columns = ['SITENAME','ID','CARBONATE'], axis = 1)
model = x
model.head()

model.describe()
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=45)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

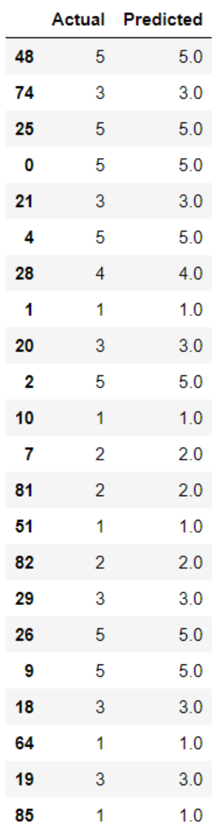
```
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
print(y_pred)

df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
y_pred)))

plt.scatter(X_test['PH'], y_test)
plt.plot(X_test['PH'], y_pred, color='red')
plt.show()
```

<u>Output: -</u>



```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 8.185371572463655e-15
Mean Squared Error: 1.2445401320920013e-28
Root Mean Squared Error: 1.1155895894512468e-14
```



	Actual	Predicted
48	5	5.0
74	3	3.0
25	5	5.0
0	5	5.0
21	3	3.0
4	5	5.0
28	4	4.0
1	1	1.0
20	3	3.0
2	5	5.0
10	1	1.0
7	2	2.0
81	2	2.0
51	1	1.0
82	2	2.0
29	3	3.0
26	5	5.0
9	5	5.0
18	3	3.0
64	1	1.0
19	3	3.0
85	1	1.0

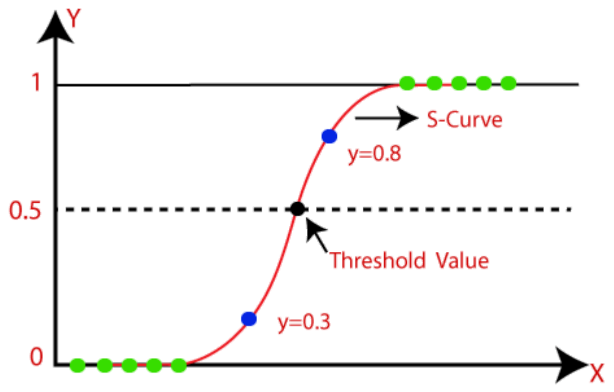
3. Logistic Regression

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis.
Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Model
Output = 0 or 1
Hypothesis => Z = WX + B

hΘ(x) = sigmoid (Z)
Sigmoid Function

Threshold value



To predict which class a data belongs to, a threshold can be set. Based upon this threshold, the obtained estimated probability is classified into classes.

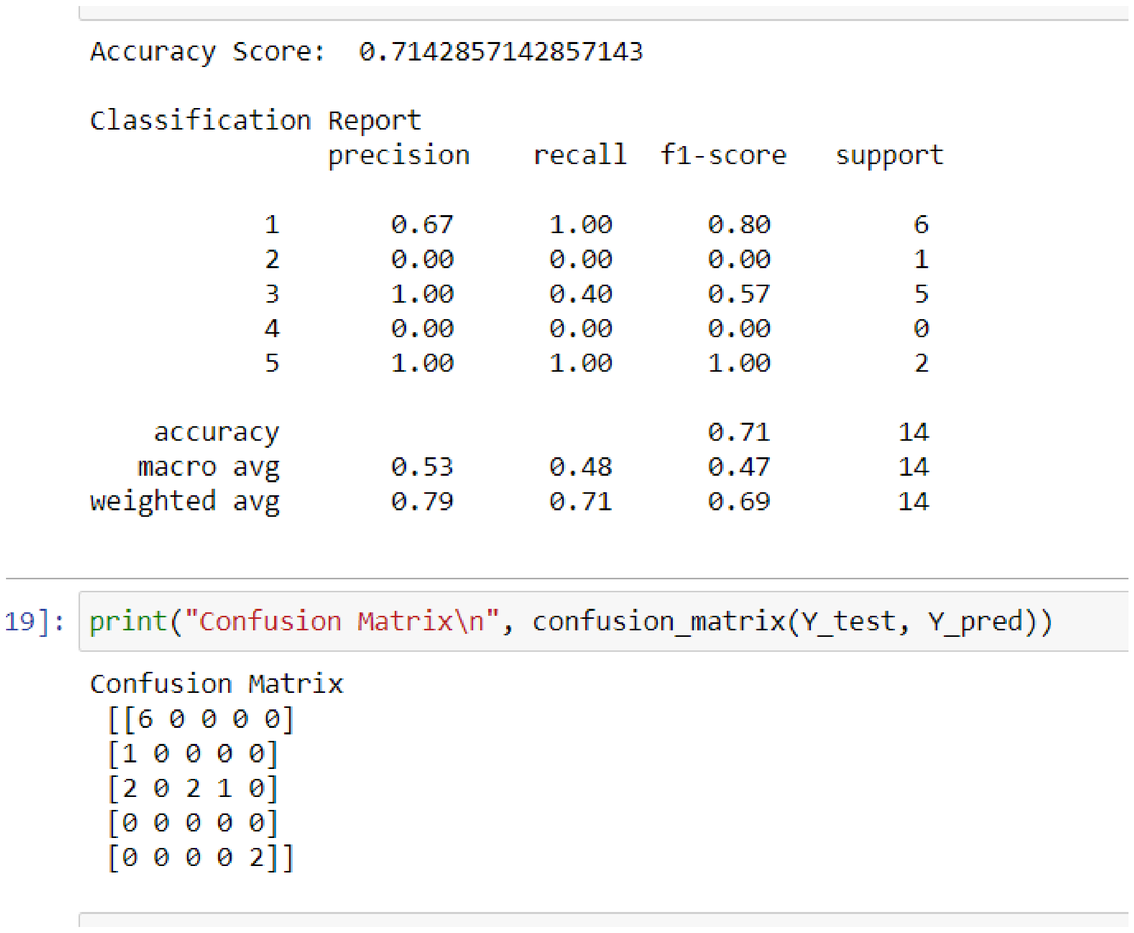
Logistic Regression Implementation using python

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings


df=pd.read_csv("dataset.csv")
df.head()

#dataset description
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
df.info()

df['waterQuality'] = df['waterQuality'].replace(['A'],'1')
df['waterQuality'] = df['waterQuality'].replace(['B'],'2')
df['waterQuality'] = df['waterQuality'].replace(['C'],'3')
df['waterQuality'] = df['waterQuality'].replace(['D'],'4')
df['waterQuality'] = df['waterQuality'].replace(['E'],'5')
df.head()

y = df['waterQuality'].astype('int32')
x = df.drop(columns = ['SITENAME','ID','CARBONATE'], axis = 1)
inputs = x
inputs.head()

from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.15)
model = LogisticRegression(max_iter=1000)
model.fit(X_train,Y_train)
```

Y_pred = model.predict(X_test)
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print("Accuracy Score: ", accuracy_score(Y_test, Y_pred))
print("\nClassification Report\n", classification_report(Y_test, Y_pred))

print("Confusion Matrix\n", confusion_matrix(Y_test, Y_pred))

<u>Output: -</u>



```
Accuracy Score:  0.7142857142857143

Classification Report
               precision    recall  f1-score   support

           1       0.67      1.00      0.80         6
           2       0.00      0.00      0.00         1
           3       1.00      0.40      0.57         5
           4       0.00      0.00      0.00         0
           5       1.00      1.00      1.00         2

    accuracy                           0.71        14
   macro avg       0.53      0.48      0.47        14
weighted avg       0.79      0.71      0.69        14
```

```
19]: print("Confusion Matrix\n", confusion_matrix(Y_test, Y_pred))
```

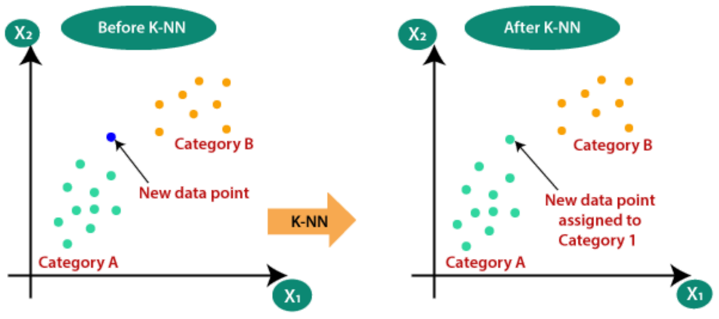
```
Confusion Matrix
 [[6 0 0 0 0]
 [1 0 0 0 0]
 [2 0 2 1 0]
 [0 0 0 0 0]
 [0 0 0 0 2]]
```

4. K Nearest Neighbor

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.
KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

The following two properties would define KNN well −

Lazy learning algorithm − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
Non-parametric learning algorithm − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data
Pros and Cons of KNN
<u>Pros</u>



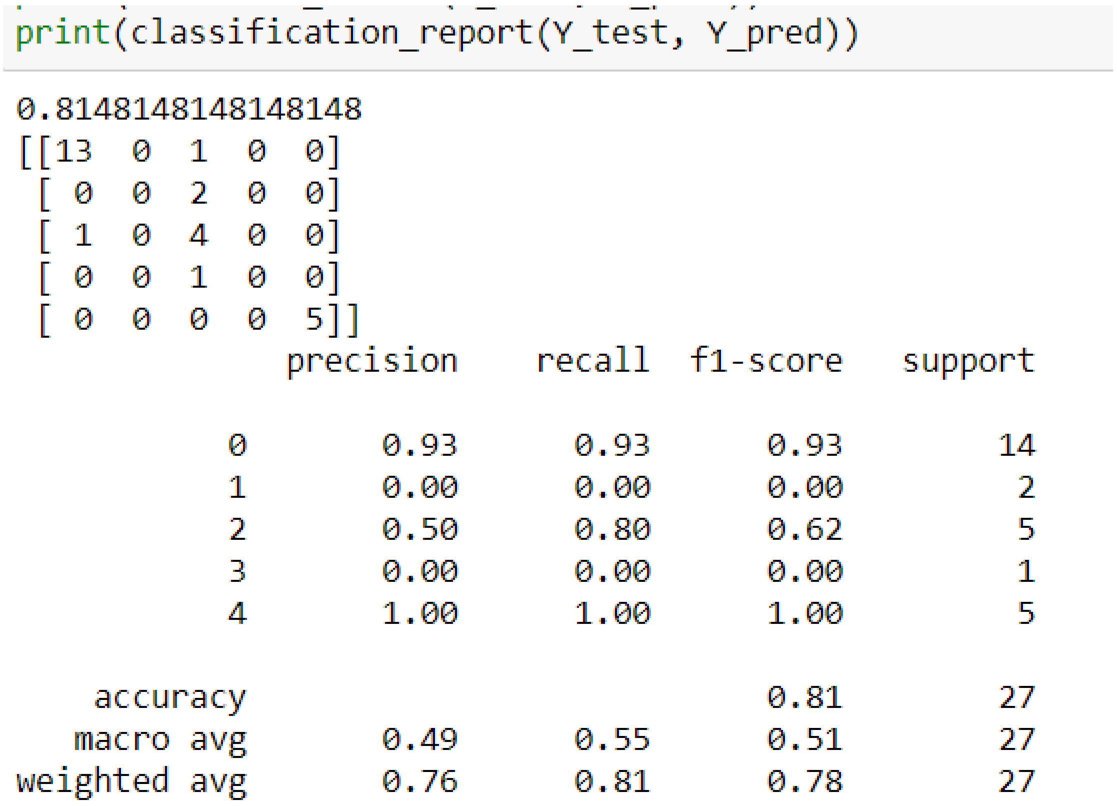
- It is very simple algorithm to understand and interpret.
- It is very useful for nonlinear data because there is no assumption about data in this algorithm.
- It is a versatile algorithm as we can use it for classification as well as regression.
- It has relatively high accuracy but there are much better supervised learning models than KNN.

<u>Cons</u>
- It is computationally a bit expensive algorithm because it stores all the training data.
- High memory storage required as compared to other supervised learning algorithms.
- Prediction is slow in case of big N.
- It is very sensitive to the scale of data as well as irrelevant features.

KNN implementation using Python

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings

df=pd.read_csv("dataset.csv")
df.head()

#dataset description
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
df.info()

df['waterQuality'] = df['waterQuality'].replace(['A'],'0')
df['waterQuality'] = df['waterQuality'].replace(['B'],'1')
df['waterQuality'] = df['waterQuality'].replace(['C'],'2')
df['waterQuality'] = df['waterQuality'].replace(['D'],'3')
df['waterQuality'] = df['waterQuality'].replace(['E'],'4')
df.head()

y = df['waterQuality'].astype('int32')
x = df.drop(columns = ['SITENAME','ID','CARBONATE'], axis = 1)
x.head()

model = x
model.head()

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

knn = KNeighborsClassifier()
```

```
Y = model['waterQuality']
X = model.drop(columns=['waterQuality'])

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=35)


from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 15)


knn.fit(X_train, Y_train)

Y_pred = knn.predict(X_test)

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings


warnings.filterwarnings('ignore')
from sklearn.metrics import classification_report, confusion_matrix
print(accuracy_score(Y_test, Y_pred))
print(confusion_matrix(Y_test, Y_pred))
print(classification_report(Y_test, Y_pred))
```



```
print(classification_report(Y_test, Y_pred))
```

```
0.8148148148148148
[[13  0  1  0  0]
 [ 0  0  2  0  0]
 [ 1  0  4  0  0]
 [ 0  0  1  0  0]
 [ 0  0  0  0  5]]
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        14
           1       0.00      0.00      0.00         2
           2       0.50      0.80      0.62         5
           3       0.00      0.00      0.00         1
           4       1.00      1.00      1.00         5

    accuracy                           0.81        27
   macro avg       0.49      0.55      0.51        27
weighted avg       0.76      0.81      0.78        27
```

5. Decision Tree

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

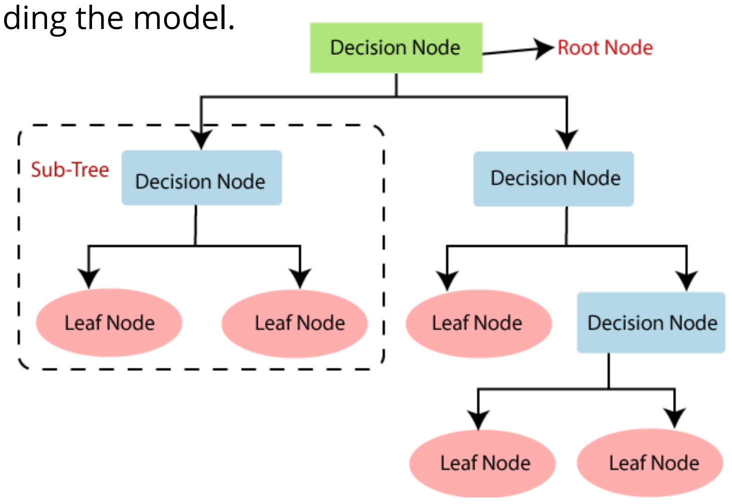
While making decision tree, at each node of tree we ask different type of questions. Based on the asked question we will calculate the information gain corresponding to it.

Information Gain

Information gain is used to decide which feature to split on at each step in building the tree. Simplicity is best, so we want to keep our tree small. To do so, at each step we should choose the split that results in the purest daughter nodes. A commonly used measure of purity is called information. For each node of the tree, the information value measures how much information a feature gives us about the class. The split with the highest information gain will be taken as the first split and the process will continue until all children nodes are pure, or until the information gain is 0.

Here are some assumptions that we made while using decision tree:

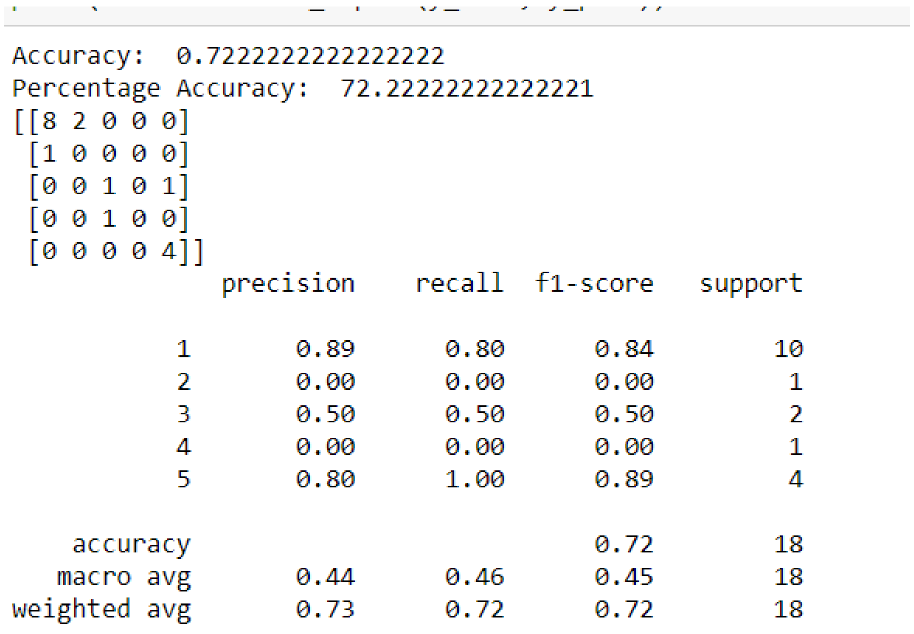
- At the beginning, we consider the whole training set as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values records are distributed recursively.
- We use statistical methods for ordering attributes as root or the internal node.



Implementation of Decision Tree using Python

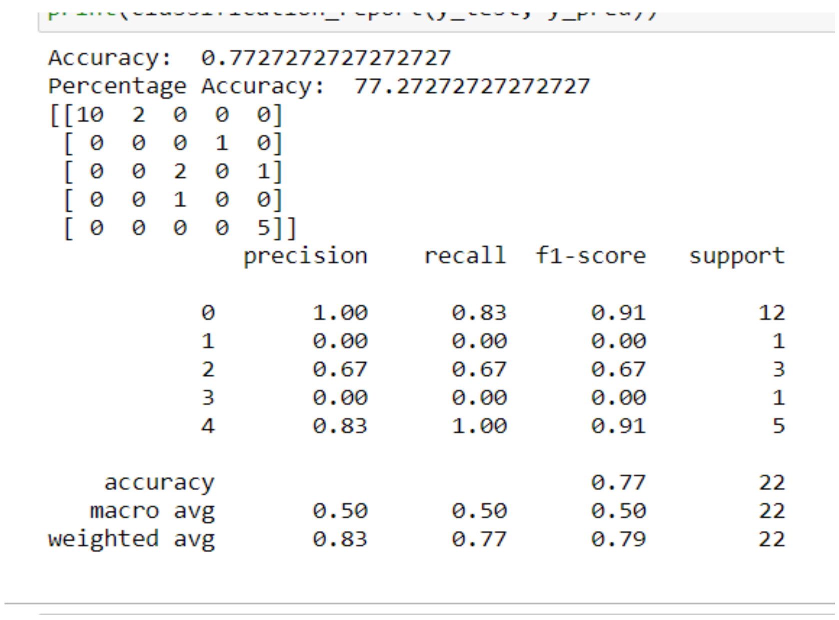
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings


df=pd.read_csv("dataset.csv")
df.head()

#dataset description
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
df.info()

df['waterQuality'] = df['waterQuality'].replace(['A'],'1')
df['waterQuality'] = df['waterQuality'].replace(['B'],'2')
df['waterQuality'] = df['waterQuality'].replace(['C'],'3')
df['waterQuality'] = df['waterQuality'].replace(['D'],'4')
df['waterQuality'] = df['waterQuality'].replace(['E'],'5')
df.head()

y = df['waterQuality'].astype('int32')
x = df.drop(columns = ['SITENAME','ID','CARBONATE'], axis = 1)
model = x
model.head()

from sklearn.preprocessing import LabelEncoder

inputs = model.drop(columns = ['waterQuality'], axis = 1)
target = model['waterQuality'].astype('int32')

le=LabelEncoder()
```

```
target=le.fit_transform(model['waterQuality'])
X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.2,random_state = 35)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=1000)
classifier.fit(X_train, y_train.ravel())
y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Percentage Accuracy: ", accuracy_score(y_test, y_pred) * 100)

warnings.filterwarnings('ignore')
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```



```
Accuracy:  0.7222222222222222
Percentage Accuracy:  72.22222222222221
[[8 2 0 0 0]
 [1 0 0 0 0]
 [0 0 1 0 1]
 [0 0 1 0 0]
 [0 0 0 0 4]]
              precision    recall  f1-score   support

           1       0.89      0.80      0.84        10
           2       0.00      0.00      0.00         1
           3       0.50      0.50      0.50         2
           4       0.00      0.00      0.00         1
           5       0.80      1.00      0.89         4

    accuracy                           0.72        18
   macro avg       0.44      0.46      0.45        18
weighted avg       0.73      0.72      0.72        18
```

Decision Tree -Gini Index



```
Accuracy:  0.7727272727272727
Percentage Accuracy:  77.27272727272727
[[10  2  0  0  0]
 [ 0  0  0  1  0]
 [ 0  0  2  0  1]
 [ 0  0  1  0  0]
 [ 0  0  0  0  5]]
              precision    recall  f1-score   support

           0       1.00      0.83      0.91        12
           1       0.00      0.00      0.00         1
           2       0.67      0.67      0.67         3
           3       0.00      0.00      0.00         1
           4       0.83      1.00      0.91         5

    accuracy                           0.77        22
   macro avg       0.50      0.50      0.50        22
weighted avg       0.83      0.77      0.79        22
```

6. Naive Bayes

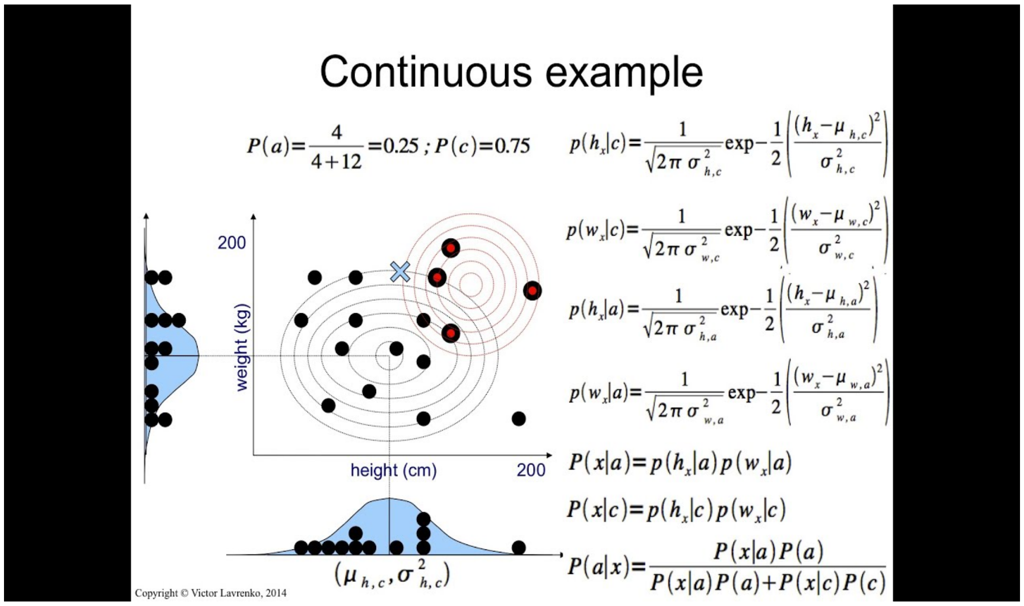
It is a classification method based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

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

Where,
P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.
P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.
P(A) is Prior Probability: Probability of a hypothesis before observing the evidence.
P(B) is Marginal Probability: Probability of Evidence.

Here the values of references are continuous so we have to use Gaussian


200
weight (kg)
height (cm)
200
$(\mu_{h,c}, \sigma^2_{h,c})$
Copyright © Victor Lavrenko, 2014

Continuous example

$$P(a) = \frac{4}{4+12} = 0.25\ ;\ P(c) = 0.75$$

$$p(h_x|c) = \frac{1}{\sqrt{2\pi\,\sigma^2_{h,c}}} \exp -\frac{1}{2}\left(\frac{(h_x - \mu_{h,c})^2}{\sigma^2_{h,c}}\right)$$

$$p(w_x|c) = \frac{1}{\sqrt{2\pi\,\sigma^2_{w,c}}} \exp -\frac{1}{2}\left(\frac{(w_x - \mu_{w,c})^2}{\sigma^2_{w,c}}\right)$$

$$p(h_x|a) = \frac{1}{\sqrt{2\pi\,\sigma^2_{h,a}}} \exp -\frac{1}{2}\left(\frac{(h_x - \mu_{h,a})^2}{\sigma^2_{h,a}}\right)$$

$$p(w_x|a) = \frac{1}{\sqrt{2\pi\,\sigma^2_{w,a}}} \exp -\frac{1}{2}\left(\frac{(w_x - \mu_{w,a})^2}{\sigma^2_{w,a}}\right)$$

$$P(x|a) = p(h_x|a)\,p(w_x|a)$$

$$P(x|c) = p(h_x|c)\,p(w_x|c)$$

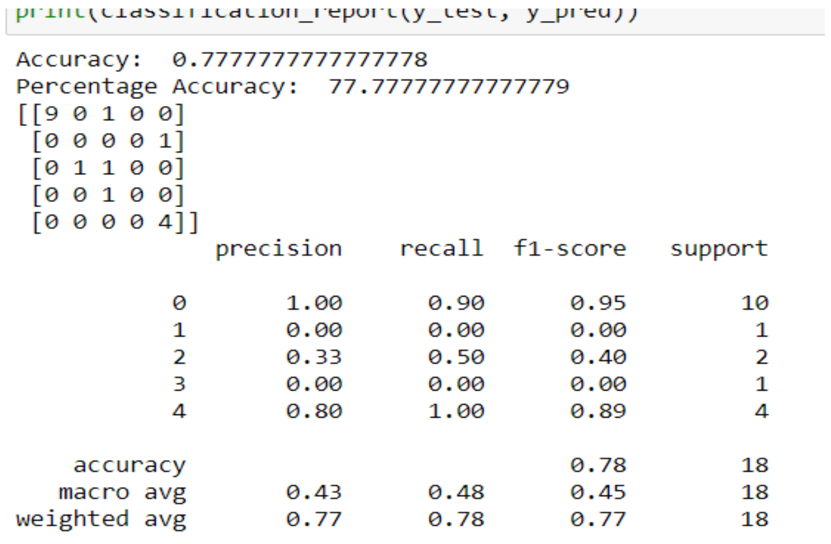
$$P(a|x) = \frac{P(x|a)P(a)}{P(x|a)P(a) + P(x|c)P(c)}$$

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

Implementation of Naive Bayes using Python

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings

df=pd.read_csv("dataset.csv")
df.head()

#dataset description
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
df.info()

df['waterQuality'] = df['waterQuality'].replace(['A'],'1')
df['waterQuality'] = df['waterQuality'].replace(['B'],'2')
df['waterQuality'] = df['waterQuality'].replace(['C'],'3')
df['waterQuality'] = df['waterQuality'].replace(['D'],'4')
df['waterQuality'] = df['waterQuality'].replace(['E'],'5')
df.head()

y = df['waterQuality'].astype('int32')
x = df.drop(columns = ['SITENAME','ID','CARBONATE'], axis = 1)
model = x
model.head()

inputs = model.drop(columns = ['waterQuality'], axis = 1)
target = model['waterQuality'].astype('int32')

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
target=le.fit_transform(model['waterQuality'])
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(inputs,target,test_size=0.2,
random_state = 35)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train.ravel())
y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Percentage Accuracy: ", accuracy_score(y_test, y_pred) * 100)

warnings.filterwarnings('ignore')
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```



```
print(classification_report(y_test, y_pred))
```

```
Accuracy:  0.7777777777777778
Percentage Accuracy:  77.77777777777779
[[9 0 1 0 0]
 [0 0 0 0 1]
 [0 1 1 0 0]
 [0 0 1 0 0]
 [0 0 0 0 4]]
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        10
           1       0.00      0.00      0.00         1
           2       0.33      0.50      0.40         2
           3       0.00      0.00      0.00         1
           4       0.80      1.00      0.89         4

    accuracy                           0.78        18
   macro avg       0.43      0.48      0.45        18
weighted avg       0.77      0.78      0.77        18
```

7. Support Vector Machine (SVM):

Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

SVM Kernels

In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM.
Linear Kernel
It can be used as a dot product between any two observations. The formula of linear kernel is as below −

$$K(x,x_i)=\mathrm{sum}(x*x_i)$$

From the above formula, we can see that the product between two vectors say x & xi is the sum of the multiplication of each pair of input values.

Polynomial Kernel

It is more generalized form of linear kernel and distinguish curved or nonlinear input space. Following is the formula for polynomial kernel −

$$k(X,Xi)=1+\mathrm{sum}(X*X_i)\text{^}d$$

Here d is the degree of polynomial, which we need to specify manually in the learning algorithm.

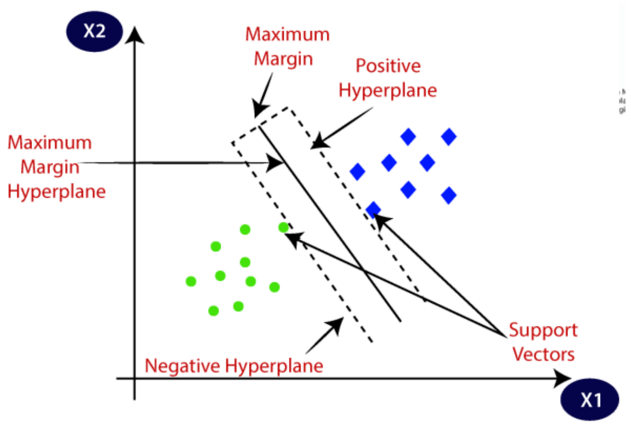
Radial Basis Function (RBF) Kernel

RBF kernel, mostly used in SVM classification, maps input space in indefinite dimensional space. Following formula explains it mathematically −

$$K(x,xi)=\exp(-gamma*sum(x-xi^2))$$

Here, gamma ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of gamma is 0.1.

As we implemented SVM for linearly separable data, we can implement it in Python for the data that is not linearly separable. It can be done by using kernels.



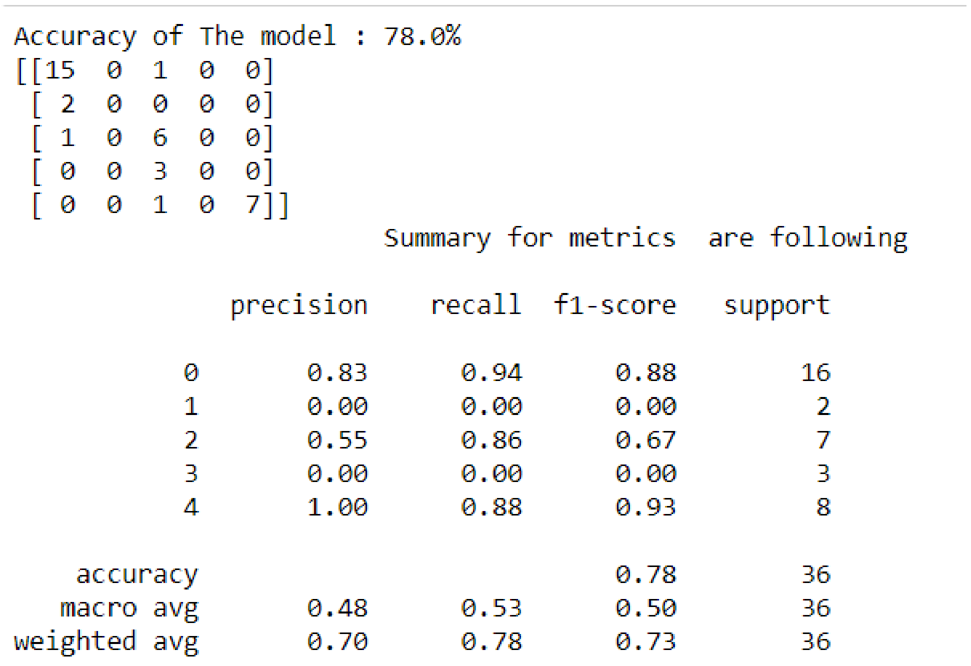
<u>Implementation of SVM using python</u>

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings


df=pd.read_csv("dataset.csv")
df.head()

#dataset description
```

```
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
df.info()

df['waterQuality'] = df['waterQuality'].replace(['A'],'1')
df['waterQuality'] = df['waterQuality'].replace(['B'],'2')
df['waterQuality'] = df['waterQuality'].replace(['C'],'3')
df['waterQuality'] = df['waterQuality'].replace(['D'],'4')
df['waterQuality'] = df['waterQuality'].replace(['E'],'5')
df.head()

y = df['waterQuality'].astype('int32')
x = df.drop(columns = ['SITENAME','ID','CARBONATE'], axis = 1)
model = x
model.head()

inputs = model.drop(columns = ['waterQuality'], axis = 1)
target = model['waterQuality'].astype('int32')


from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
target=le.fit_transform(df['waterQuality'])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(inputs,target,test_size=0.4)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn import svm, metrics
model = svm.SVC(gamma=0.04)
```

```
#fit to the trainin data
model.fit(X_train,y_train)

# now to Now predict the value of the digit on the test data
from sklearn import svm, metrics
y_pred = model.predict(X_test)
acc = metrics.accuracy_score(y_pred,y_test)
print("Accuracy of The model : %s"%(acc*100).round(0)+"%")
from sklearn import svm, metrics
import warnings
warnings.filterwarnings('ignore')

warnings.filterwarnings('ignore')
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print("\t\t\tSummary for metrics  are following\n  \n%s\n"
    % (metrics.classification_report(y_test, y_pred)))
```

<u>Output: -</u>



```
Accuracy of The model : 78.0%
[[15  0  1  0  0]
 [ 2  0  0  0  0]
 [ 1  0  6  0  0]
 [ 0  0  3  0  0]
 [ 0  0  1  0  7]]
                        Summary for metrics  are following

              precision    recall  f1-score   support

           0       0.83      0.94      0.88        16
           1       0.00      0.00      0.00         2
           2       0.55      0.86      0.67         7
           3       0.00      0.00      0.00         3
           4       1.00      0.88      0.93         8

    accuracy                           0.78        36
   macro avg       0.48      0.53      0.50        36
weighted avg       0.70      0.78      0.73        36
```

8. Results: -

Sl.No	Algorithm	Accuracy(%)
1	Linear Regression	**100**
2	Logistic Regression	**71.4**
3	KNN	**81**
4	Decision Tree- Info Gain	**72.22**
5	Decision Tree- Gini index	**77.27**
6	Naive Bayes	**77.77**
7	SVM	**78**