IDS PROJECT REPORT

Applying ML Classification Algorithm - Predicting Quality of Wine based on physicochemical tests of Wine.

Team Members

Durvesh Umesh Sangle (20UCS068)
Garvit Hinger (20UCS070)
Manav Jangid (20UCS111)
Vasu Sachdeva (20UCS224)

GitHub Repository

https://github.com/manavjangid5/QuadSquad_IDS_Project

Course Instructors

Dr. Subrat K. Dash
Dr. Sakthi Balan
Dr. Aloke Datta

Department of Computer Science Engineering The LNM Institute of Information Technology

Problem Set:

You have been provided with a CSV dataset that has 12 columns and 1599 rows.
Our task is to analyze the dataset and The goal is to model wine quality based on physicochemical tests by developing a supervised machine learning model.

We need to collect a dataset from the given website and perform the following steps:
1. Data pre-processing and its visualization
2. Explain all the inferences we got from our data.
3. Explain what ML Classification Algorithms are being used.
4. Reason for implementing those algorithms and their implementation.
5. Output the result of the testing set and its visualization.

All of the tasks are performed with the help of pre-existing Python Libraries such as:

- sklearn.metrics
- matplotlib
- train_test_split
- numpy
- pandas

Attributes:

- **fixed acidity**
- **volatile acidity**
- **citric acid**
- **residual sugar**
- **Chlorides**
- **free sulfur dioxide**
- **total sulfur dioxide**
- **density**
- **pH**
- **sulphates**
- **alcohol**

Implementation:

1. Importing Libraries and Loading Dataset:

1) Importing Libraries:

Basically we have to first import the library

• **pandas:** for manipulation and analysis of data.

• **numpy:** contains a large collection of high-level mathematical functions to operate on large arrays and matrices with multiple dimensions.

• **matplotlib:** for embedding plots into applications using general-purpose GUI toolkits.

• **sklearn.metrics:** implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

• **train_test_split:** used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train, X_test, y_train, and y_test. X_train and y_train sets are used for training and fitting the model.

2) Reading Data Set:

a)

```
7    df = pandas.read_csv("winequality-red.csv",delimiter=';')
8    df.head()
```

```
PS E:\LNMIIT Courses\Sem 5\IDS\IDS Project> & C:/Python310/python.exe "e:/LNMIIT Courses/Sem 5/IDS/IDS Project/ids.py"
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  ...  density    pH  sulphates  alcohol  quality
0            7.4              0.70         0.00             1.9      0.076  ...   0.9978  3.51       0.56      9.4        5
1            7.8              0.88         0.00             2.6      0.098  ...   0.9968  3.20       0.68      9.8        5
2            7.8              0.76         0.04             2.3      0.092  ...   0.9970  3.26       0.65      9.8        5
3           11.2              0.28         0.56             1.9      0.075  ...   0.9980  3.16       0.58      9.8        6
4            7.4              0.70         0.00             1.9      0.076  ...   0.9978  3.51       0.56      9.4        5
```

b) Overall Description of the dataset

```
PS E:\LNMIIT Courses\Sem 5\IDS\IDS Project> & C:/Python310/python.exe "e:/LNMIIT Courses/Sem 5/IDS/IDS Project/ids.py"
       fixed acidity  volatile acidity  citric acid  residual sugar    chlorides  ...      density           pH    sulphates      alcohol      quality
count    1599.000000       1599.000000  1599.000000     1599.000000  1599.000000  ...  1599.000000  1599.000000  1599.000000  1599.000000  1599.000000
mean        8.319637          0.527821     0.270976        2.538806     0.087467  ...     0.996747     3.311113     0.658149    10.422983     5.636023
std         1.741096          0.179060     0.194801        1.409928     0.047065  ...     0.001887     0.154386     0.169507     1.065668     0.807569
min         4.600000          0.120000     0.000000        0.900000     0.012000  ...     0.990070     2.740000     0.330000     8.400000     3.000000
25%         7.100000          0.390000     0.090000        1.900000     0.070000  ...     0.995600     3.210000     0.550000     9.500000     5.000000
50%         7.900000          0.520000     0.260000        2.200000     0.079000  ...     0.996750     3.310000     0.620000    10.200000     6.000000
75%         9.200000          0.640000     0.420000        2.600000     0.090000  ...     0.997835     3.400000     0.730000    11.100000     6.000000
max        15.900000          1.580000     1.000000       15.500000     0.611000  ...     1.003690     4.010000     2.000000    14.900000     8.000000
```

2. Data Exploration:

Distribution of Data on the basis of different Categorical Attributes:

A) For Volatile Acidity

Range	Count	Percentage
[0,0.2)	17	1.06%
[0.2,0.4)	391	24.45%
[0.4,0.6)	640	40.03%
[0.6,0.8)	444	27.77%
[0.8,1)	83	5.19%
[1,1.2)	20	1.25%
[1.2,1.4)	3	0.19%
[1.4,1.6]	1	0.06%

B) For Fixed Acidity

Range	Count	Percentage
[4, 5)	3	0.19%
[5, 6)	55	3.44%
[6, 7)	254	15.88%
[7, 8)	504	31.52%
[8,9)	316	19.76%
[9, 10)	191	11.94%
[10, 11)	138	8.63%
[11, 12)	72	4.50%
[12, 13)	43	2.69%
[13, 14)	14	0.88%
[14, 15)	2	0.13%
[15, 16)	7	0.44%

C) For Residual Sugar

Range	Count	Percentage
[0,4)	1463	91.49%
[4,8)	115	7.19%
[8,12)	13	0.81%
[12,16]	8	0.50%

D) For Citric Acid

Range	Count	Percentage
[0,0.2)	606	37.90%
[0.2,0.4)	521	32.58%
[0.4,0.6)	385	24.08%
[0.6,0.8)	86	5.38%
[0.8,1)	1	0.06%

E) For Chlorides

Range	Count	Percentage
[0,0.1)	1363	85.24%
[0.1,0.2)	194	12.13%
[0.2,0.3)	20	1.25%
[0.3,0.4)	9	0.56%
[0.4,0.5)	11	0.69%
[0.5,0.6)	0	0.00%
[0.6,0.7]	2	0.13%

F) For Free Sulfur Dioxide

Range	Count	Percentage
(0,10]	605	34.78%
(10,20]	555	34.71%
(20,30]	276	17.26%
(30,40]	122	7.63%
(40,50]	25	1.56%
(50,60]	12	0.75%
(60,70]	3	0.19%
(70,80]	1	0.06%

G) For Total Sulfur Dioxide

Range	Count	Percentage
[0,25)	510	31.89%
[25,50)	543	33.96%
[50,75)	272	17.01%
[75,100)	147	9.19%
[100,125)	77	4.82%
[125,150)	41	2.56%
[150,300)	9	0.56%

H) For Density

Range	Count	Percentage
(0,1.0]	1528	95.56%
(1.0,2.0)	71	4.44%

I) For pH

Range	Count	Percentage
[0,3)	35	2.19%
[3,3.25)	515	32.21%
[3.25,3.5)	881	55.10%
[3.5,3.75)	161	10.07%
[3.75,4)	5	0.31%
[4,4.25)	2	0.13%

J) For Sulphate

Range	Count	Percentage
(0,0.5)	151	9.44%
[0.5,1)	1389	86.87%
[1,1.5)	51	3.19%
[1.5,2)	8	0.50%

K) For Alcohol

Range	Count	Percentage
[8,9)	7	0.44%
[9,10)	673	42.09%
[10,11)	452	28.27%
[11,12)	305	19.07%
[12,13)	133	8.32%
[13,14)	21	1.31%
[14,15)	8	0.50%

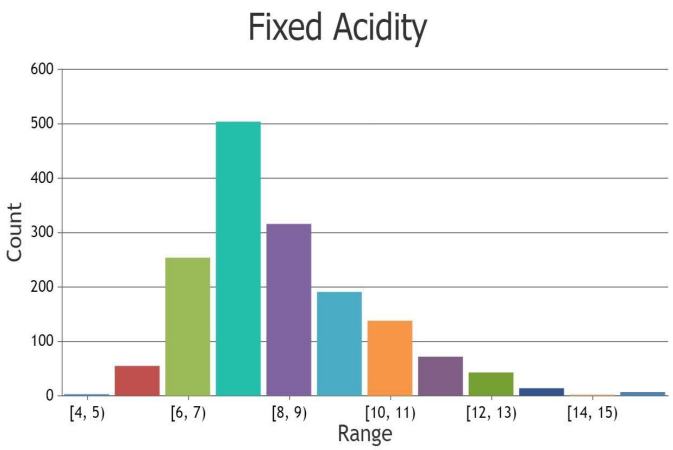
L) For Quality

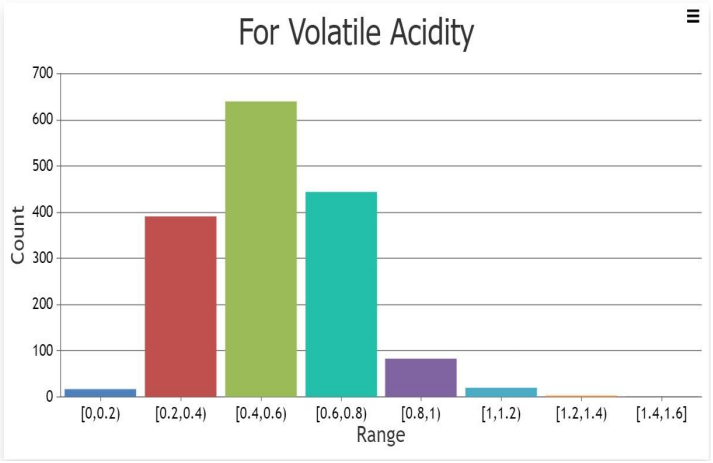
Value	Count	Percentage
3	10	0.63%
4	53	3.31%
5	681	42.59%
6	638	39.90%
7	199	12.45%
8	18	1.13%

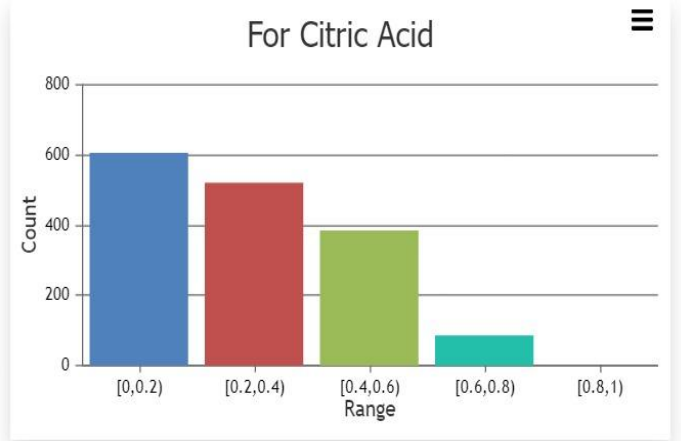
3. Data Visualization:

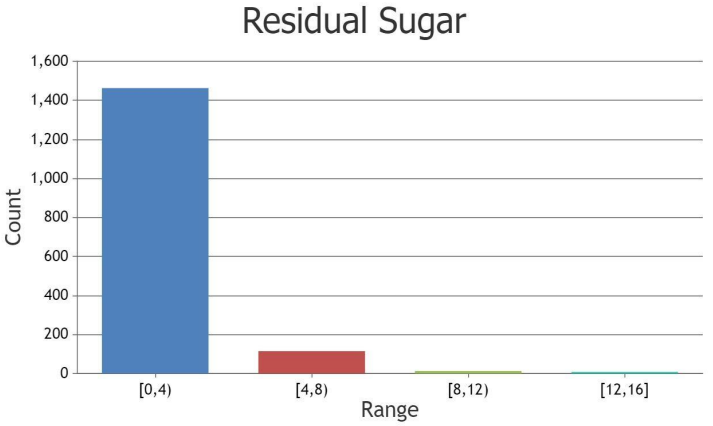
Here we would try to find out a relation between each column of the final numerical dataset and the target attribute.

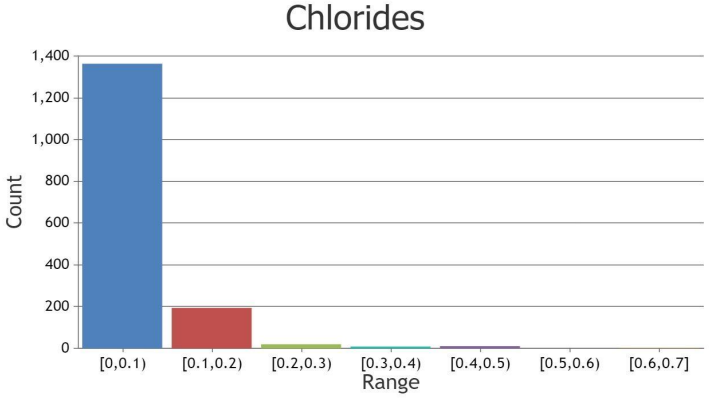
Bar chart visualization for attributes: (count[no.of values in that range] VS Range of values)

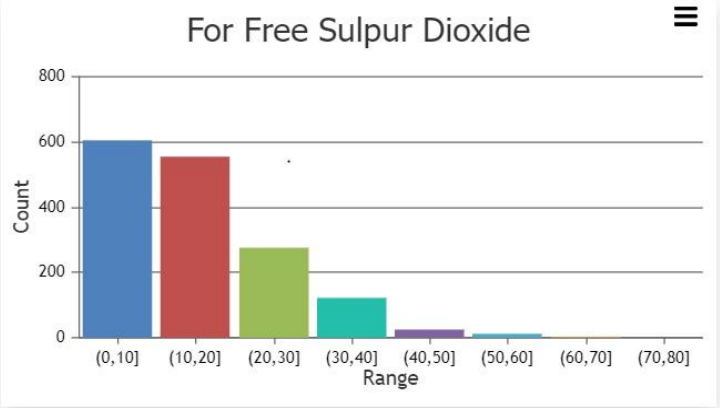


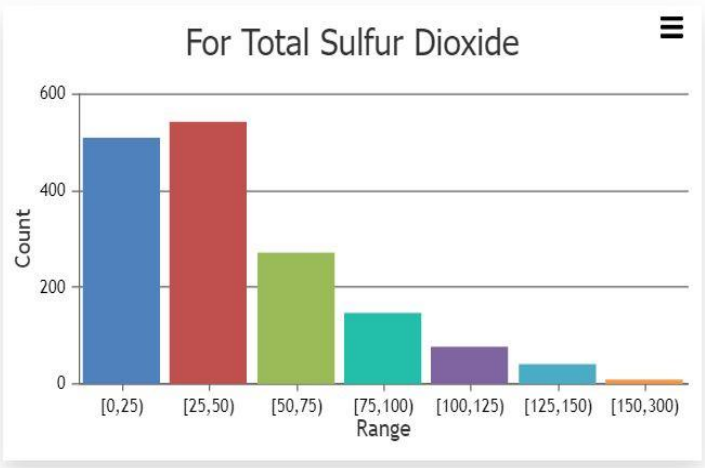


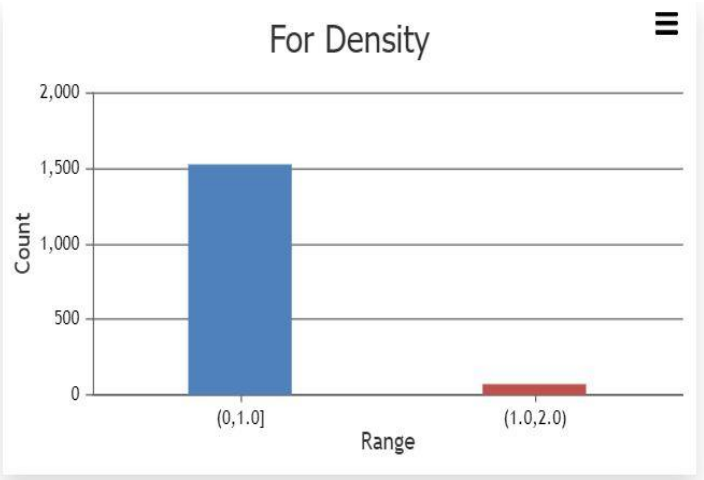


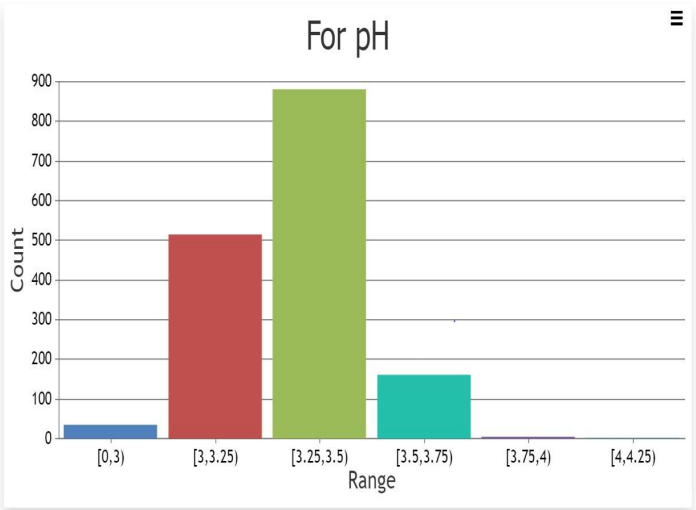


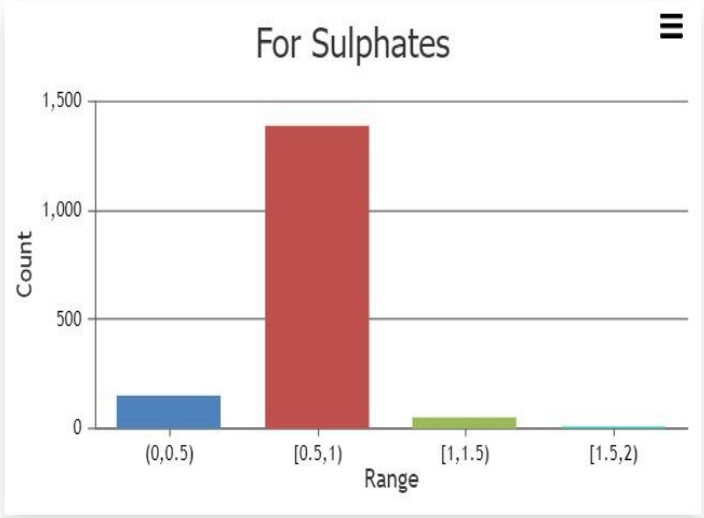


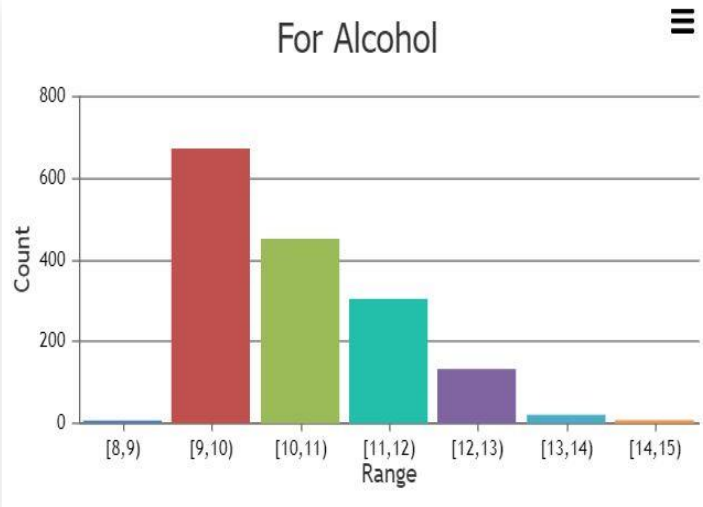


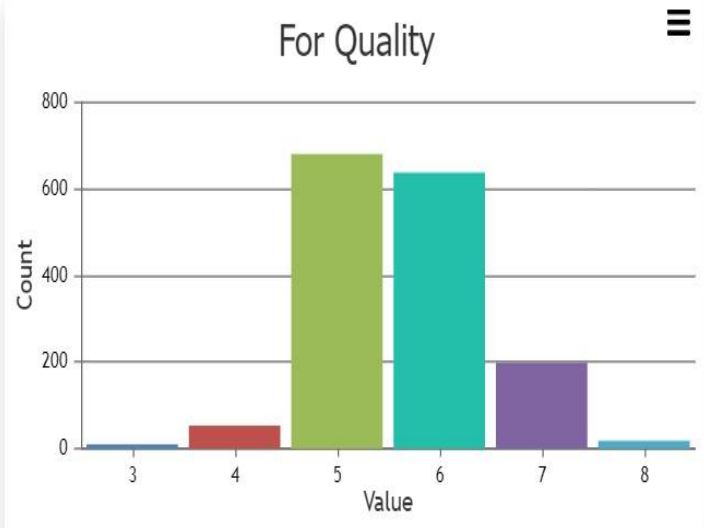

For Total Sulfur Dioxide
600
400
200
0
Count
[0,25)
[25,50)
[50,75)
[75,100)
[100,125)
[125,150)
[150,300)
Range


For Density
2,000
1,500
1,000
500
0
Count
(0,1.0]
(1.0,2.0)
Range


For pH
900
800
700
600
500
400
300
200
100
0
Count
[0,3)
[3,3.25)
[3.25,3.5)
[3.5,3.75)
[3.75,4)
[4,4.25)
Range


For Sulphates
1,500
1,000
500
0
Count
(0,0.5)
[0.5,1)
[1,1.5)
[1.5,2)
Range


For Alcohol
800
600
400
200
0
Count
[8,9)
[9,10)
[10,11)
[11,12)
[12,13)
[13,14)
[14,15)
Range


For Quality
800
600
400
200
0
Count
3
4
5
6
7
8
Value

4. ML CLASSIFICATION ALGORITHMS:

● After getting a cleaned dataset, we can now apply our prediction algorithms, to predict the quality of our wine.
● In our project, instead of applying just one, we use 9 different classification algorithms to predict the results.
● The motivation to apply all these algorithms was that we wanted to compare their accuracy results to see which algorithm works better on our dataset.

We applied the algorithms given below.

- ○ Logistic regression
- ○ K-Nearest Neighborhood
- ○ Decision Tree Classifier
- ○ Random Forest Classifier
- ○ Naive Bayes Classifier
  - Gaussian Naive Bayes
  - Multinomial Naive Bayes
- ○ Support Vector Machine
  - Polynomial
  - Radial Basis Function
  - Linear

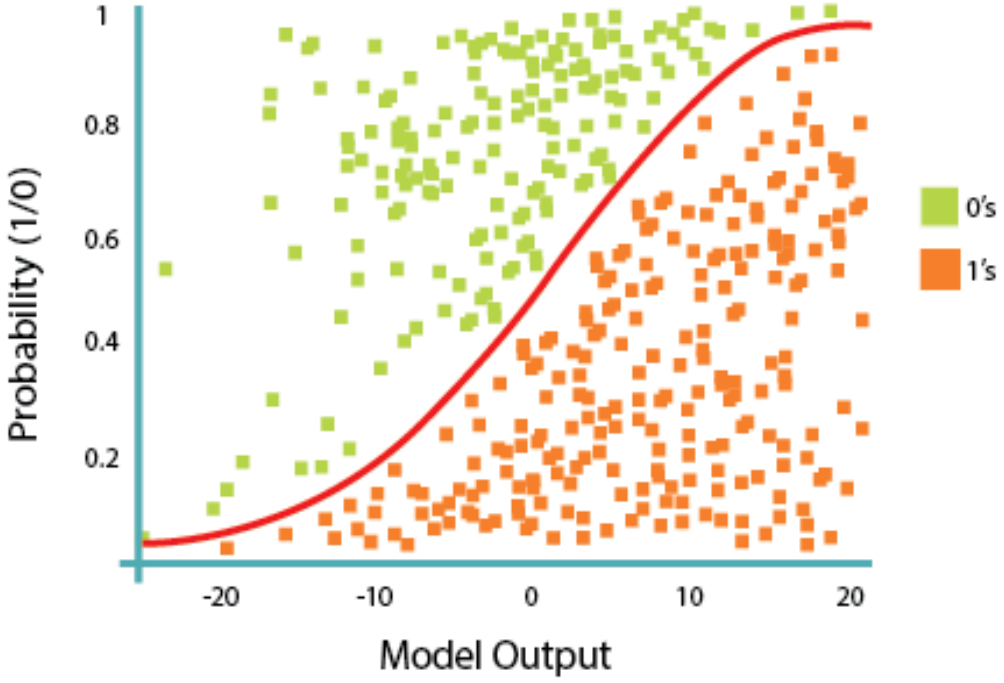
● For each case, we've visualized the Confusion Matrix along with it.
● We've also displayed the accuracy percentage for each case too.

1. **Logistic Regression:**

   Logistic regression is one of the simplest and most widely used supervised machine learning algorithms for category classification. The basic concept of logistic regression is easy to understand and can be used as a **basic algorithm for binary (0 or 1) classification problems.**

   This model is used to calculate the probability of a particular class. It can also be used for multiclass attribute values. It basically follows a linear regression model, but the continuous output values are passed through a function called the "**sigmoid function**", which is used to **scale the values between 0 and 1.**

   (Example for Logistic Regression performance)



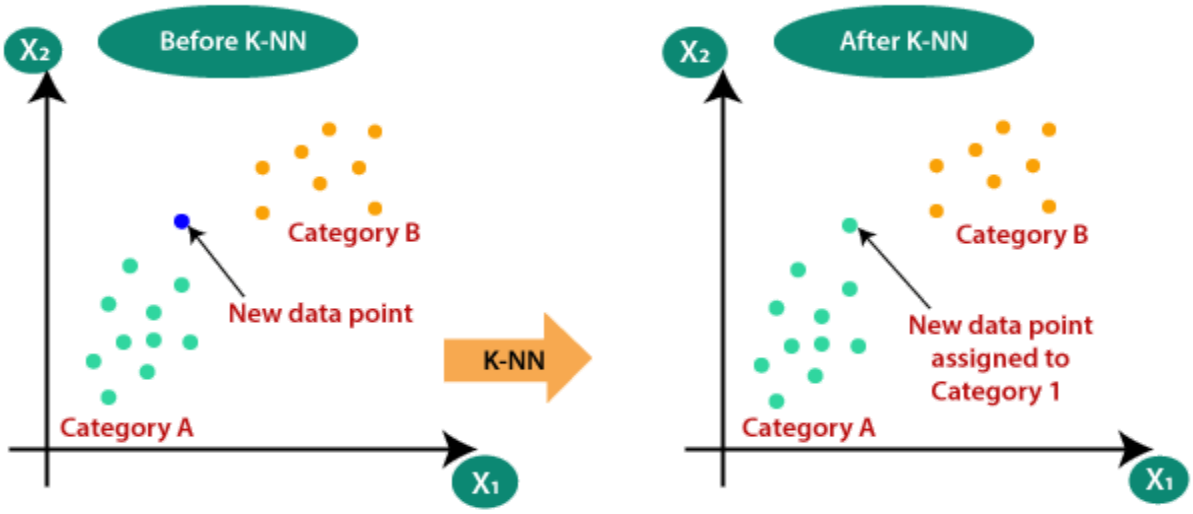
2. K-Nearest Neighbor algorithm:

Mainly used for classification problems, K-NN algorithms assume similarities between new cases/data and available cases and assign new cases to the category that is most similar to the available categories.

Instead of learning from the training set immediately, save the data set and perform actions on the data set during classification.

Reasons for choosing K-NN is suppose we have two categories: category A and category B, and we have a new data point x1, so this data point will be in any of those categories. A K-NN algorithm is needed to solve this kind of problem. With the help of K-NN, we can easily identify the category or class of a particular record. Consider the following illustration.

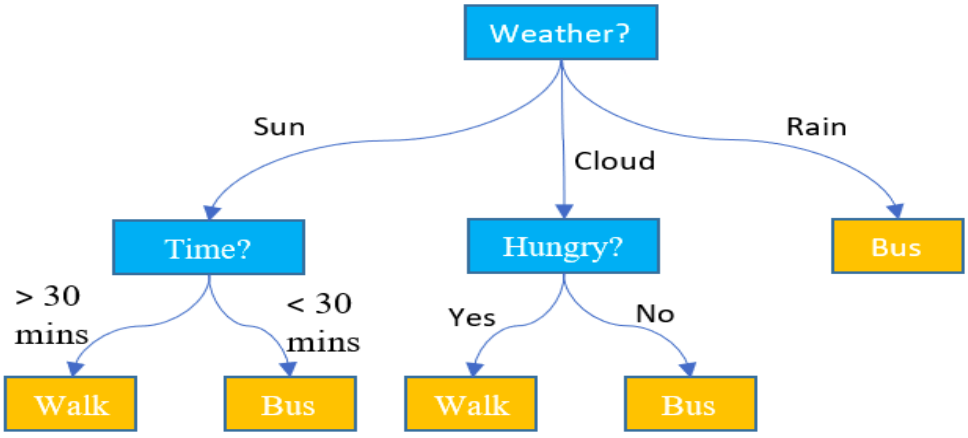
(Example for K-Nearest Neighbor performance)



3. **Decision Tree Classifier:**

   This model builds a decision tree based on the input data set. Useful for predicting classes for new input datasets. This is like a tree structure, with each node representing a condition for a attribute value. Based on the conditional output, we choose the path (response to the condition) and continue predicting class (output the class variable) until we reach a leaf node. Decision trees were built using the GINI index, entropy calculations, etc., and learned to measure impurities to determine what is considered the best split.

   (Example for Decision Tree performance)



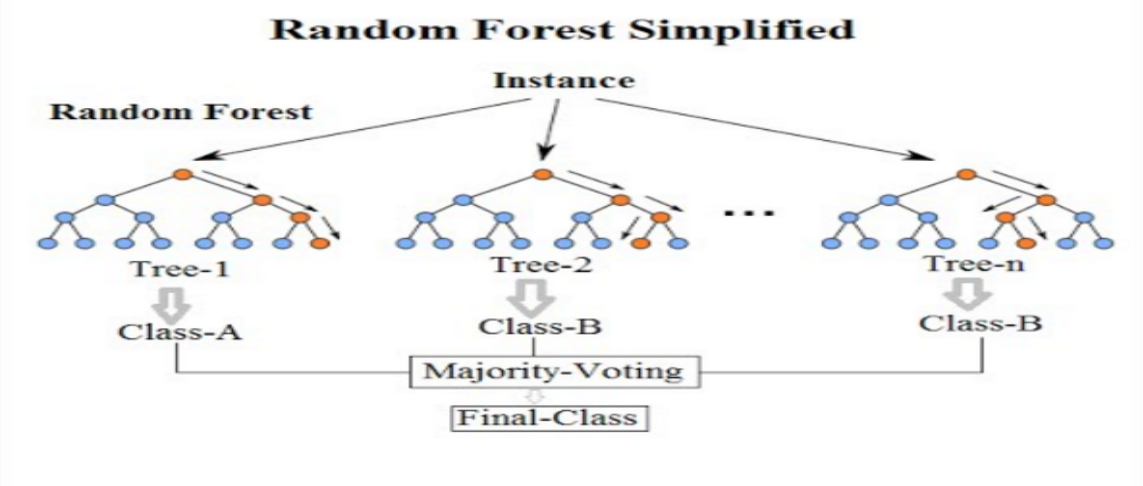
4. **Random Forest Classifier:**

   This classifier follows ensemble learning. Instead of one, multiple decision trees act as an "**ensemble**". Add randomness to the ensemble by randomly creating a forest of decision trees. Each decision tree produces an output and the final

classes of the input data set are created by majority vote.

More accurate than a single decision tree, but takes longer to train.

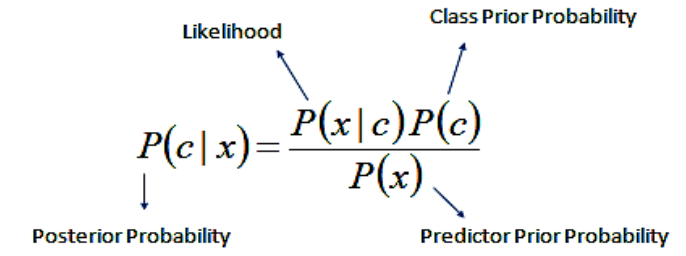
(Example for Random Forest performance)



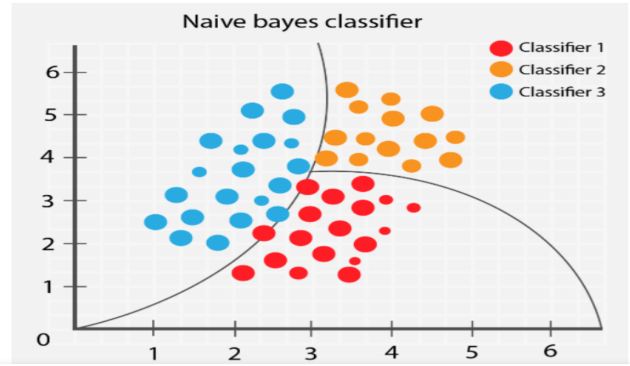
5. **Naive Bayes Classifier:**

   The naive Bayes classifier is based on Bayes' theorem, which we study with probabilities. In ML, independence is assumed among the attributes X(i) when class prediction is made. Formula to calculate probability using Bayes' theorem.

   (Example for Naive Bayes performance)



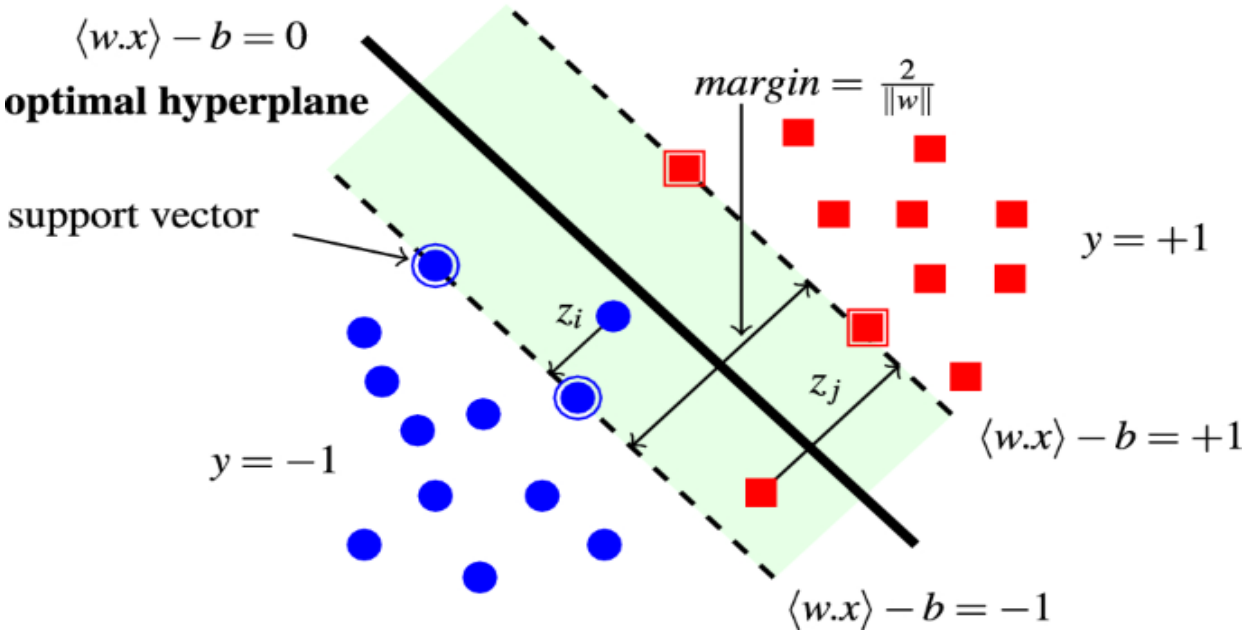
$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$



6. **Support Vector Machine:**

   The support vector machine uses the supervised learning algorithm. It is used to find the hyperplane separating data classes. Currently, there may be many hyperplanes that can separate the data, but SVM tries to find the best line for this separation. This classifier generally works well when there are clear boundaries between classes and the dataset is not large enough.

   (Example for Support Vector Machine performance)

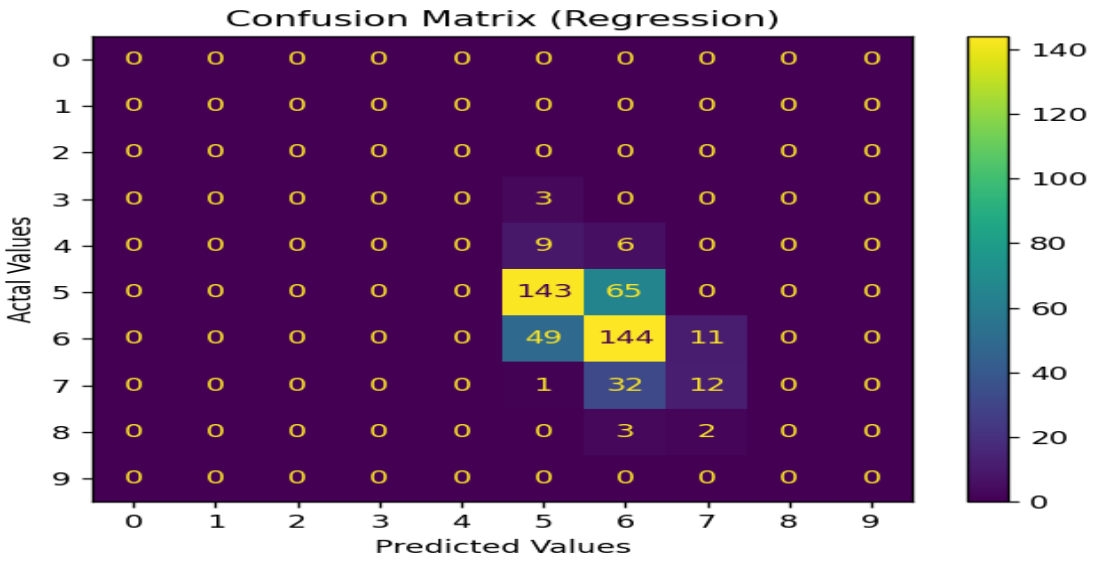


5. IMPLEMENTATION OF THE ALGORITHMS AND THEIR CONFUSION MATRIX:

1. Logistic Regression: *Accuracy: 49%*

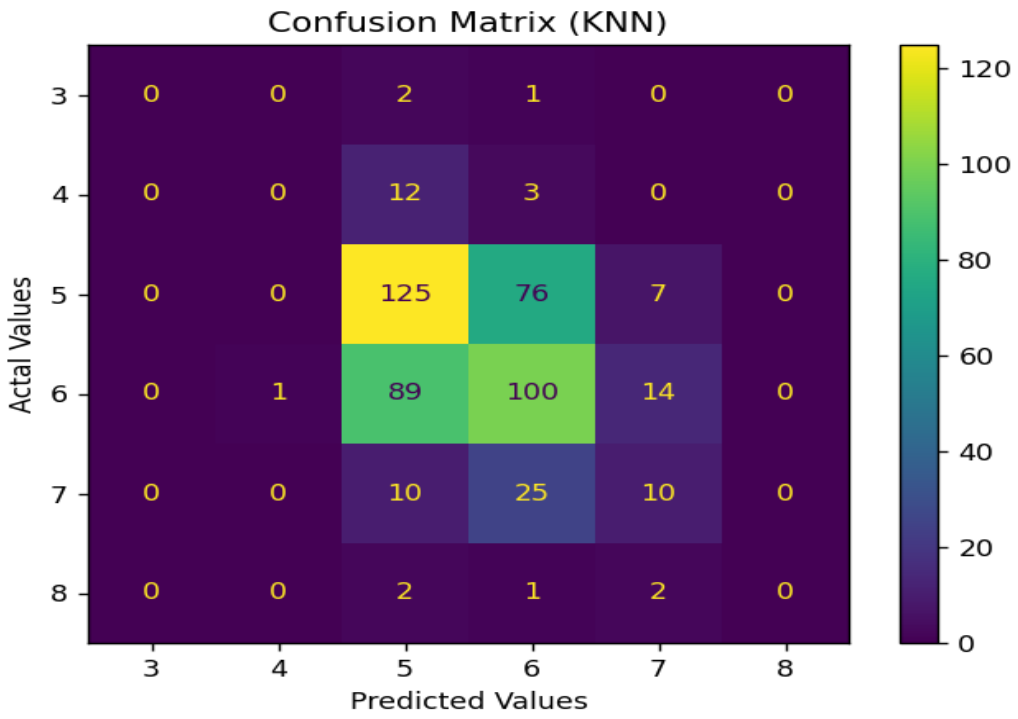
```
############################ Linear Regression ################################
from sklearn import linear_model

regr = linear_model.LinearRegression()
regr.fit(X_train.values,Y_train)

Y_pred = regr.predict(X_test.values)
print("------------------Linear Regression---------------")
y_pred1 = []
for val in Y_pred:
    y_pred1.append(round(val))

Y_pred1 = numpy.array(y_pred1)

print(Y_pred1)
```

```
#Creating confusion matrix
cm = confusion_matrix(Y_test, Y_pred1, labels=[0,1,2,3,4,5,6,7,8,9])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1,2,3,4,5,6,7,8,9])

#Plotting the confusion matrix
disp.plot()
plt.title('Confusion Matrix (Regression)')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()

print('Accuracy: %.3f' % accuracy_score(Y_test, Y_pred1))
```



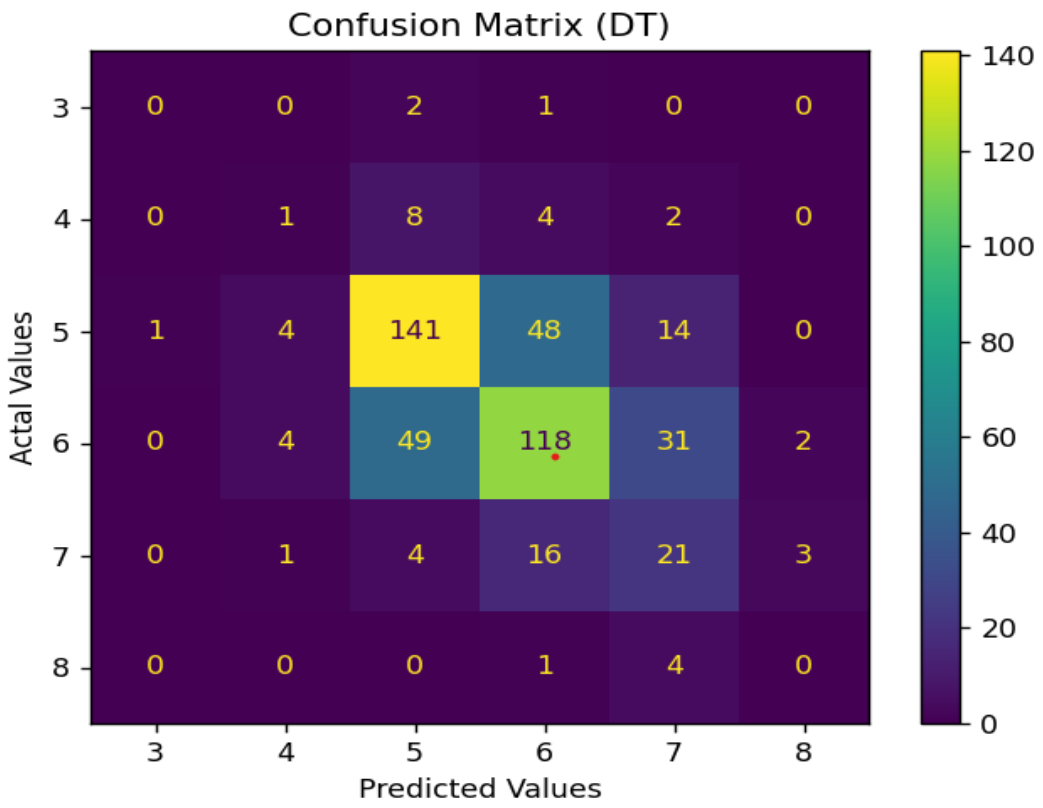
2. K-Nearest Neighbor algorithm: *Accuracy: 62.3%*

```
47  ############################ KNN ########################################
48
49  from sklearn.neighbors import KNeighborsClassifier
50  knn = KNeighborsClassifier(n_neighbors=7)
51
52  knn.fit(X_train.values, Y_train)
53
54  Y_pred_knn = knn.predict(X_test.values)
55  print("-------------KNN------------")
56  print(Y_pred_knn)
57
58  #Creating confusion matrix
59  cm = confusion_matrix(Y_test, Y_pred_knn, labels=knn.classes_)
60  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=knn.classes_)
61
62  #Plotting the confusion matrix
63  disp.plot()
64  plt.title('Confusion Matrix (KNN)')
65  plt.ylabel('Actal Values')
66  plt.xlabel('Predicted Values')
67  plt.show()
68
69  print('Accuracy: %.3f' % accuracy_score(Y_test, Y_pred_knn))
70
```



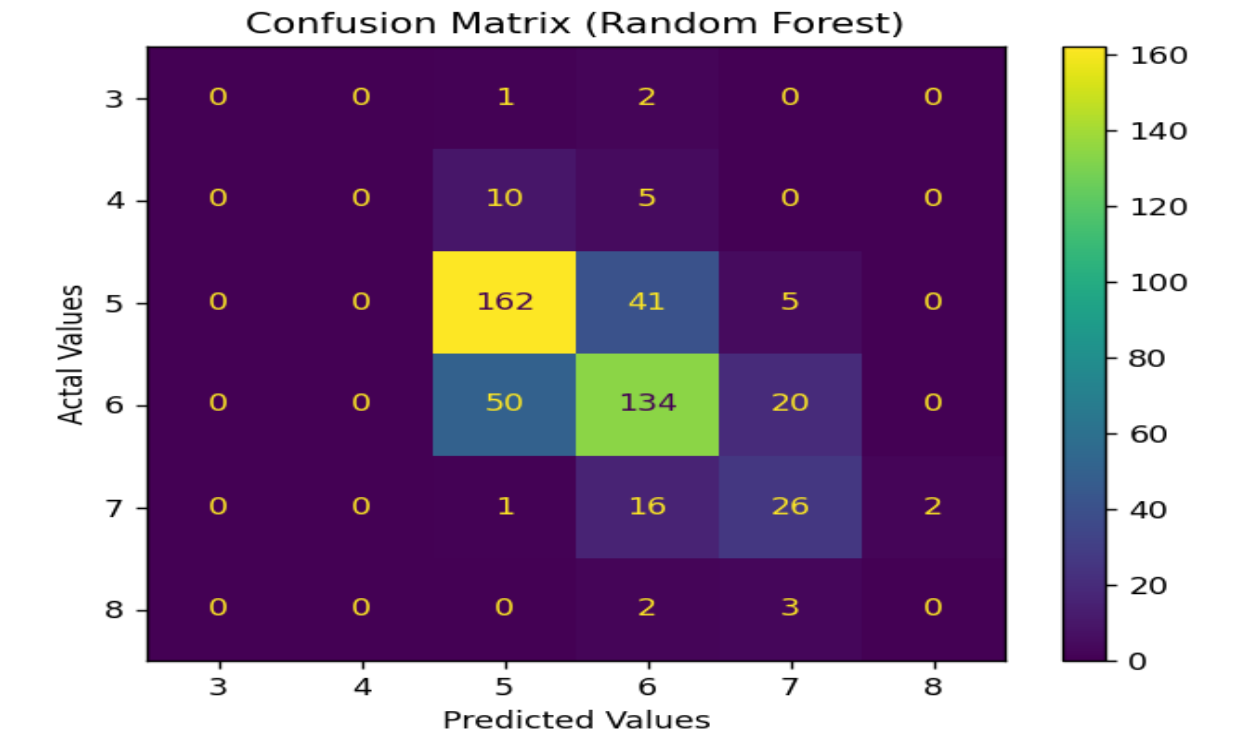
3. Decision Tree Classifier: *Accuracy: 59.0%*

```
71  ############################ DT ##########################################
72
73  from sklearn.tree import DecisionTreeClassifier
74  dtree = DecisionTreeClassifier()
75
76  dtree.fit(X_train.values, Y_train)
77
78  Y_pred_dt = dtree.predict(X_test.values)
79
80  #Creating confusion matrix
81  cm = confusion_matrix(Y_test, Y_pred_dt, labels=dtree.classes_)
82  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=dtree.classes_)
83
84  #Plotting the confusion matrix
85  disp.plot()
86  plt.title('Confusion Matrix (DT)')
87  plt.ylabel('Actal Values')
88  plt.xlabel('Predicted Values')
89  plt.show()
90
91  print("--------------DT-------------")
92  print(Y_pred_dt)
93
94  print('Accuracy: %.3f' % accuracy_score(Y_test, Y_pred_dt))
95
```



4. Random Forest Classifier: *Accuracy: 69.2%*

```
97  ############################ Random Forest ##########################################
98
99  from sklearn.ensemble import RandomForestClassifier
100 rForest = RandomForestClassifier()
101
102 rForest.fit(X_train.values, Y_train)
103
104 Y_pred_rForest = rForest.predict(X_test.values)
105
106 #Creating confusion matrix
107 cm = confusion_matrix(Y_test, Y_pred_rForest, labels=rForest.classes_)
108 disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=rForest.classes_)
109
110 #Plotting the confusion matrix
111 disp.plot()
112 plt.title('Confusion Matrix (Random Forest)')
113 plt.ylabel('Actal Values')
114 plt.xlabel('Predicted Values')
115 plt.show()
116
117 print("--------------Random Forest-------------")
118 print(Y_pred_rForest)
119
120 print('Accuracy: %.3f' % accuracy_score(Y_test, Y_pred_rForest))
121
```

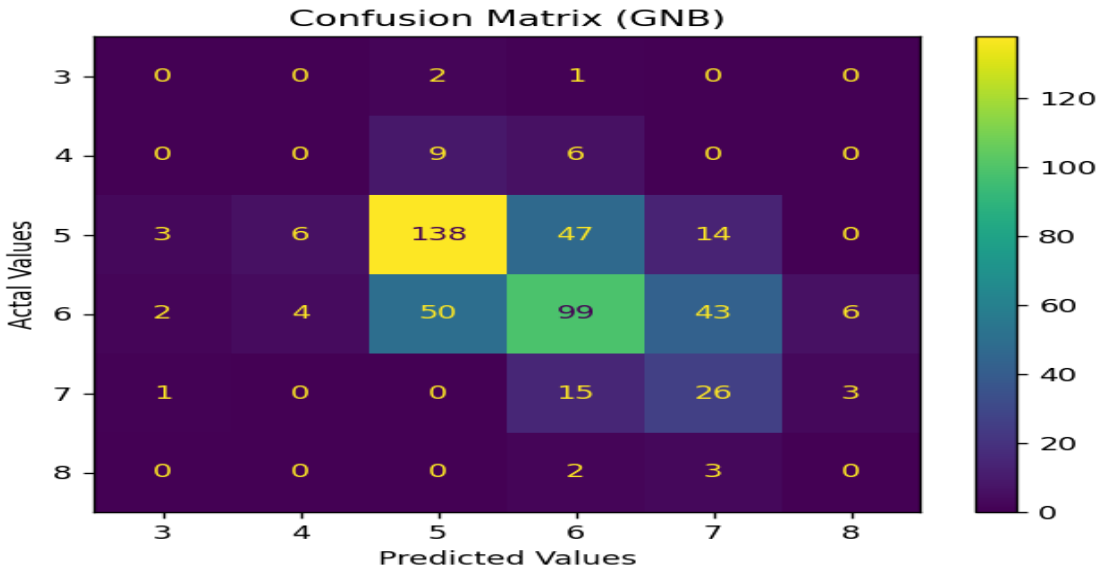


5. Naive Bayes Classifier:

```
124  ############################ Naive Bayes ##########################################
125
126  from sklearn.naive_bayes import GaussianNB
127  from sklearn.naive_bayes import MultinomialNB
128
129  gnb=GaussianNB()
130  mnb=MultinomialNB()
131  gnb.fit(X_train.values, Y_train)
132  mnb.fit(X_train.values, Y_train)
133
134  Y_pred_gnb = gnb.predict(X_test.values)
135  Y_pred_mnb = mnb.predict(X_test.values)
```

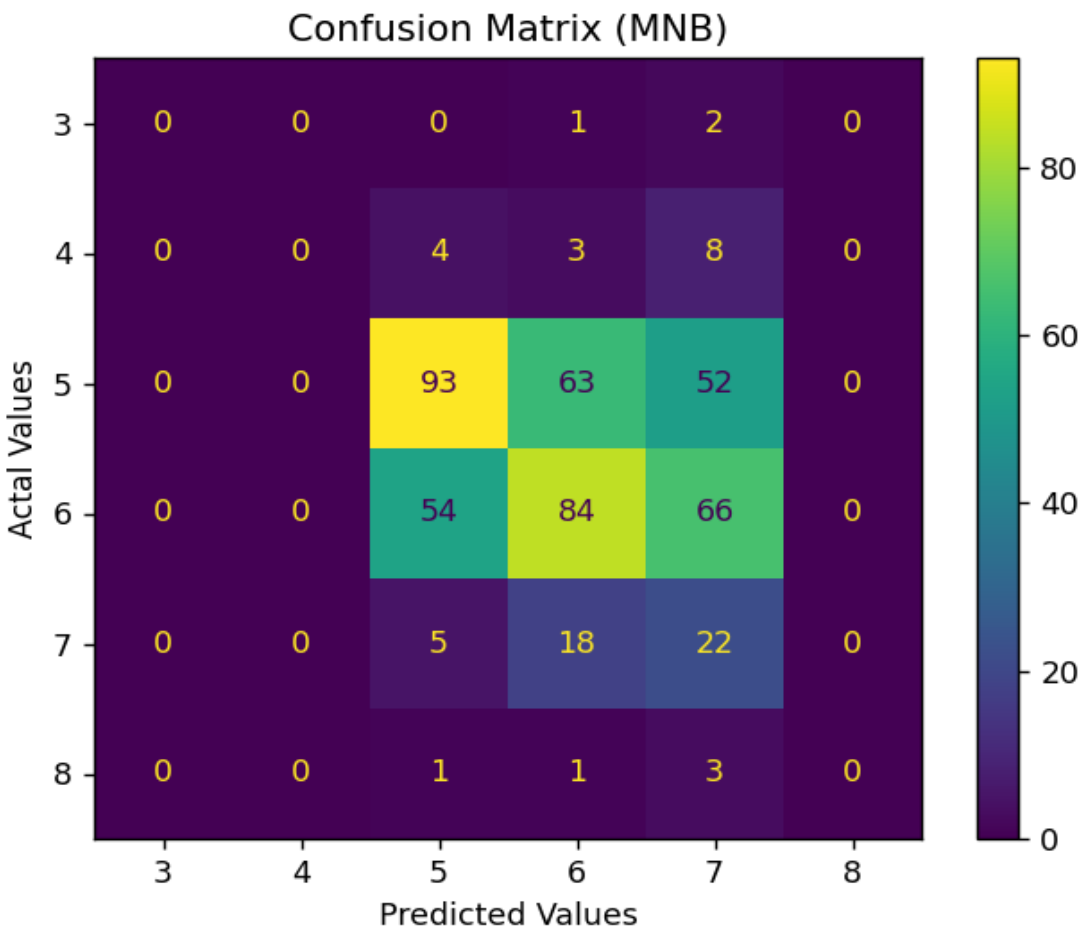
—-------GNB (Gaussian Naive Bayes) *Accuracy: 54.8%*

```
137  #Creating confusion matrix
138  cm = confusion_matrix(Y_test, Y_pred_gnb, labels=gnb.classes_)
139  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=gnb.classes_)
140
141  #Plotting the confusion matrix
142  disp.plot()
143  plt.title('Confusion Matrix (GNB)')
144  plt.ylabel('Actal Values')
145  plt.xlabel('Predicted Values')
146  plt.show()
147
148  print("-------------GNB-------------")
149  print(Y_pred_gnb)
150
151  print('Accuracy: %.3f' % accuracy_score(Y_test, Y_pred_gnb))
152
```



—------ MNB (Multinomial Naive Bayes) *Accuracy: 41.5%*

```
153
154  #Creating confusion matrix
155  cm = confusion_matrix(Y_test, Y_pred_mnb, labels=mnb.classes_)
156  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=mnb.classes_)
157
158  #Plotting the confusion matrix
159  disp.plot()
160  plt.title('Confusion Matrix (MNB)')
161  plt.ylabel('Actal Values')
162  plt.xlabel('Predicted Values')
163  plt.show()
164
165  print("--------------MNB-------------")
166  print(Y_pred_mnb)
167
168  print('Accuracy: %.3f' % accuracy_score(Y_test, Y_pred_mnb))
169
```

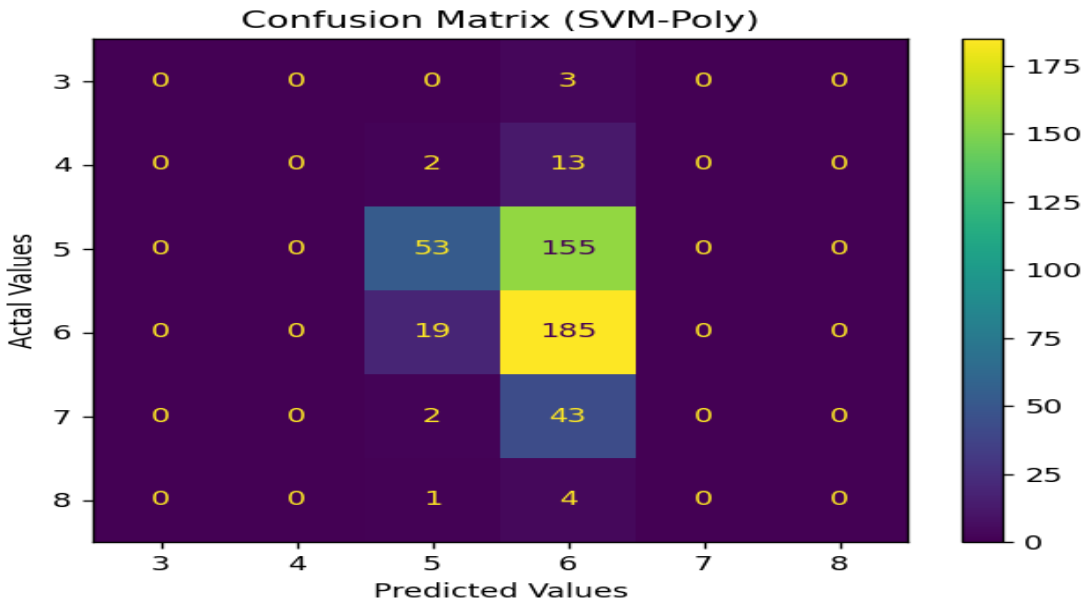


6. Support Vector Machine:

```
171  ############################ SVM ##############################################
172
173  from sklearn import svm
174  rbf = svm.SVC(kernel='rbf', gamma=0.7, C=1.0)
175  rbf.fit(X_train, Y_train)
176  poly = svm.SVC(kernel='poly', degree=3, C=1.0)
177  poly.fit(X_train, Y_train)
178  svc = svm.SVC(kernel='linear', C=1.0)
179  svc.fit(X_train, Y_train)
180  poly_pred = poly.predict(X_test)
181  rbf_pred = rbf.predict(X_test)
182  svc_pred = svc.predict(X_test)
183
```

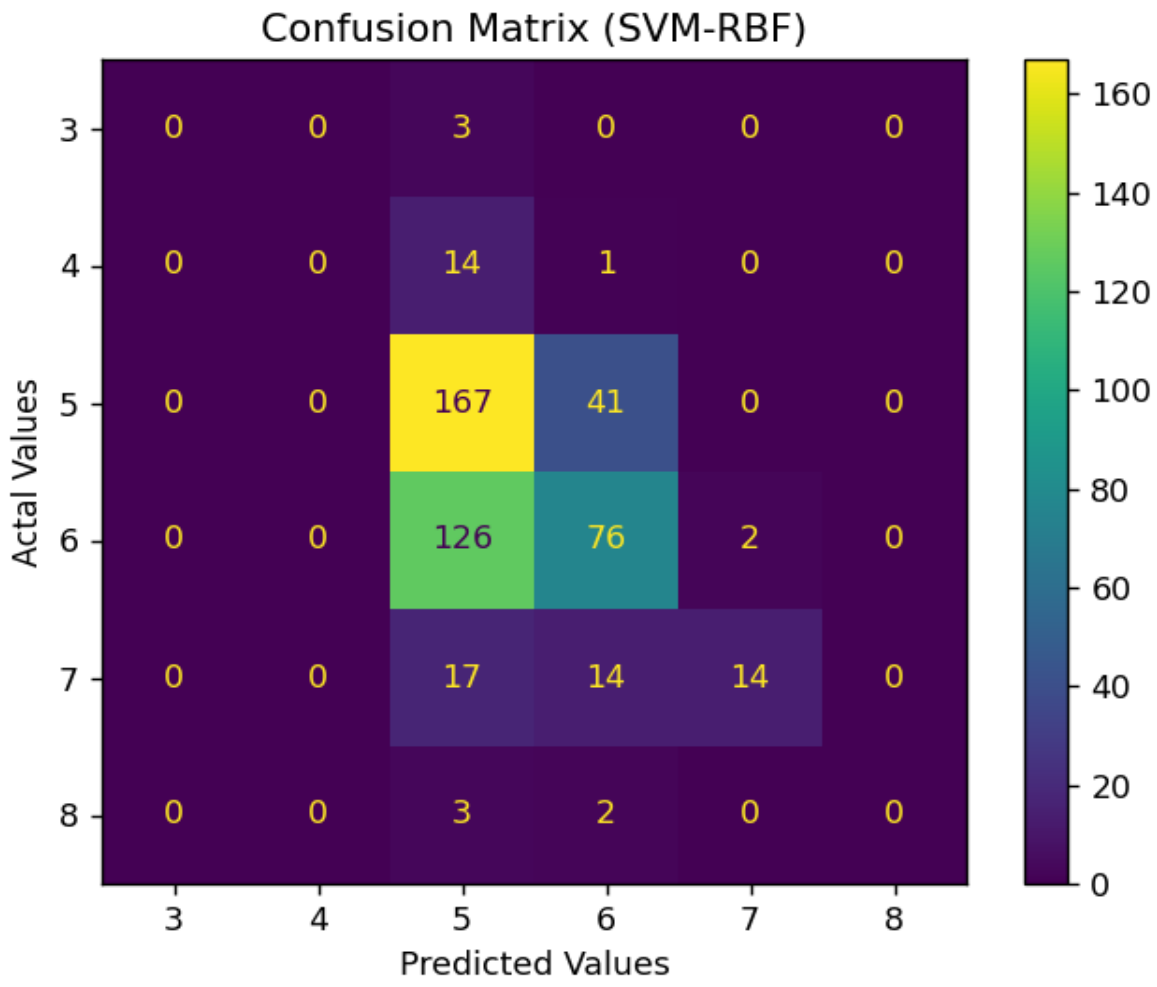
—------- Poly SVM *Accuracy: 49.6%*

```
184  #Creating confusion matrix
185  cm = confusion_matrix(Y_test, poly_pred, labels=poly.classes_)
186  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=poly.classes_)
187
188  #Plotting the confusion matrix
189  disp.plot()
190  plt.title('Confusion Matrix (SVM-Poly)')
191  plt.ylabel('Actal Values')
192  plt.xlabel('Predicted Values')
193  plt.show()
194
195  print("--------------POLY-------------")
196  print(poly_pred)
197
198  print('Accuracy: %.3f' % accuracy_score(Y_test, poly_pred))
199
```



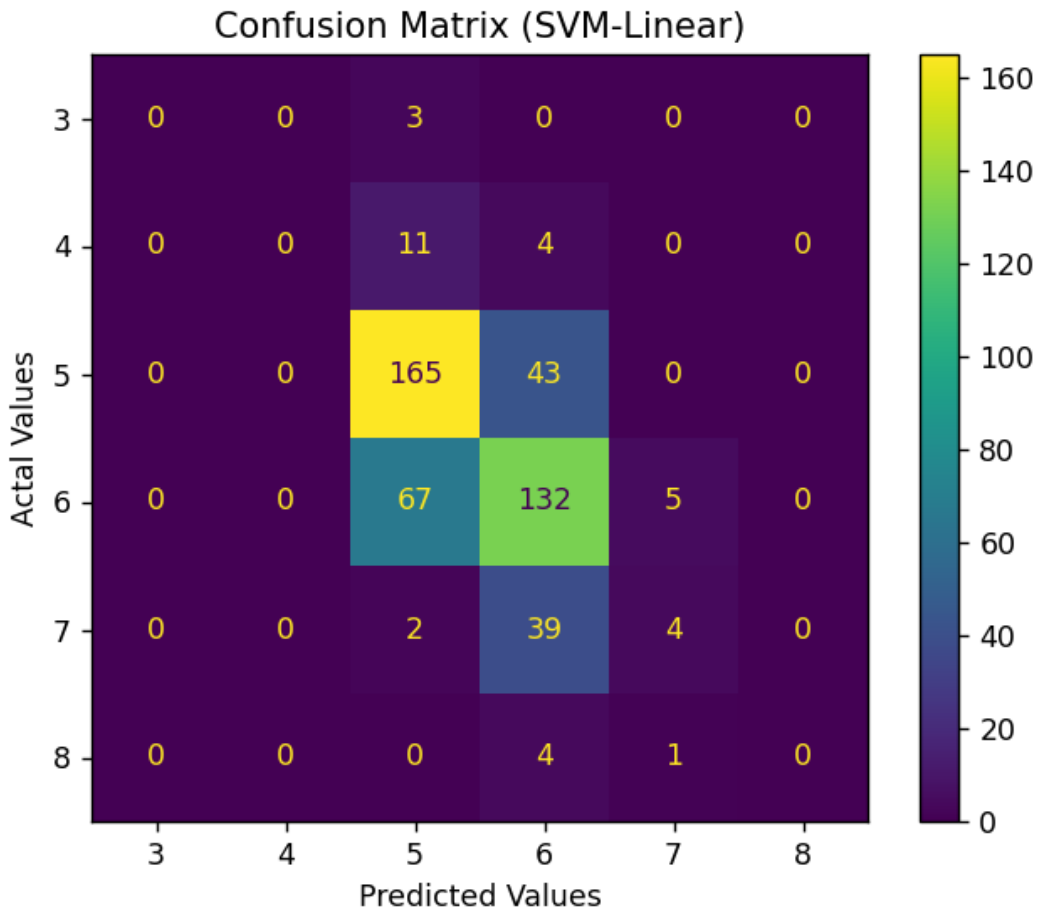
—------ RBF (Radial Basis Function SVM) *Accuracy: 53.5%*

```
201  ###################################RBF#########################################
202
203  #Creating confusion matrix
204  cm = confusion_matrix(Y_test, rbf_pred, labels=rbf.classes_)
205  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=rbf.classes_)
206
207  #Plotting the confusion matrix
208  disp.plot()
209  plt.title('Confusion Matrix (SVM-RBF)')
210  plt.ylabel('Actal Values')
211  plt.xlabel('Predicted Values')
212  plt.show()
213
214  print("--------------RBF-------------")
215  print(rbf_pred)
216
217  print('Accuracy: %.3f' % accuracy_score(Y_test, rbf_pred))
218
```



—------- SVC(Support Vector Classifier) *Accuracy: 62.7%*

```
220  ######################################### Linear ##############################
221
222  #Creating confusion matrix
223  cm = confusion_matrix(Y_test, svc_pred, labels=svc.classes_)
224  disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=svc.classes_)
225
226  #Plotting the confusion matrix
227  disp.plot()
228  plt.title('Confusion Matrix (SVM-Linear)')
229  plt.ylabel('Actal Values')
230  plt.xlabel('Predicted Values')
231  plt.show()
232
233  print("--------------Linear-------------")
234  print(svc_pred)
235
236  print('Accuracy: %.3f' % accuracy_score(Y_test, svc_pred))
237
```



6. CONCLUSION:

To summarize our results from the implementations, we get the following table:

Algorithms	**Accuracy**
Linear Regression	0.623
K-nearest Neighbor	0.490
Decision Tree	0.590
Random Forest	0.692
Naive Bayes (GBN)	0.548
Naive Bayes (MBN)	0.415
Support Vector Machine (Poly)	0.496
Support Vector Machine (RBF)	0.535
Support Vector Machine (Linear)	0.627

From the results, it's clear that the **Random Forest Algorithm** gives the highest accuracy of **0.692 (69.2%)** on our dataset.

REFERENCES

1) https://scikit-learn.org/stable/ (Classification Algorithms)
2) https://archive.ics.uci.edu/ml/datasets/Wine+Quality (Wine Dataset)
3) Tom Mitchell. Machine Learning. 1st edition, McGraw Hill, 1997.
4) https://matplotlib.org/ (To Plot Confusion Matrix)
5) https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html – Read CSV (Pandas)