DWDM_Project

June 9, 2021

1 Prediction of Quality for Different Type of Wine based on Different Feature Sets Using Supervised Machine Learning Techniques

[2]:
```
import numpy as np
import pandas as pd
import matplotlib as plt
import matplotlib.pyplot as plot
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import cross_val_score
```

[3]:
```
wine=pd.read_csv("C:\\Users\\arai2\\Desktop\\winequality.csv")
```

2 Data Analysis of Wine Content Using Pandas

[4]:
```
wine.head()
```

[4]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
```

```
3      9.8        6
4      9.4        5
```

[5]:
```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

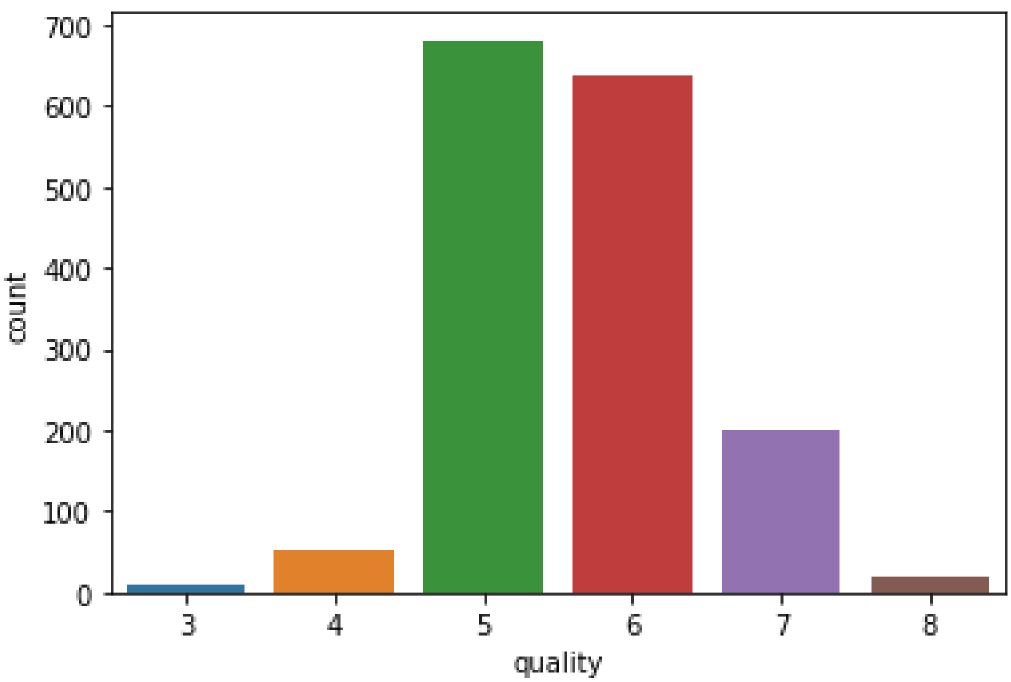
[6]:
```
wine.describe()
```

[6]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
```

```
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

[7]:
```
sns.countplot(wine['quality']) #number of wines wrt specific rating
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x9ee5d30>
```



Majority of wines fall at the score of '5'

[8]:
```
print(wine.isna().sum()) #To check the number of null values
```

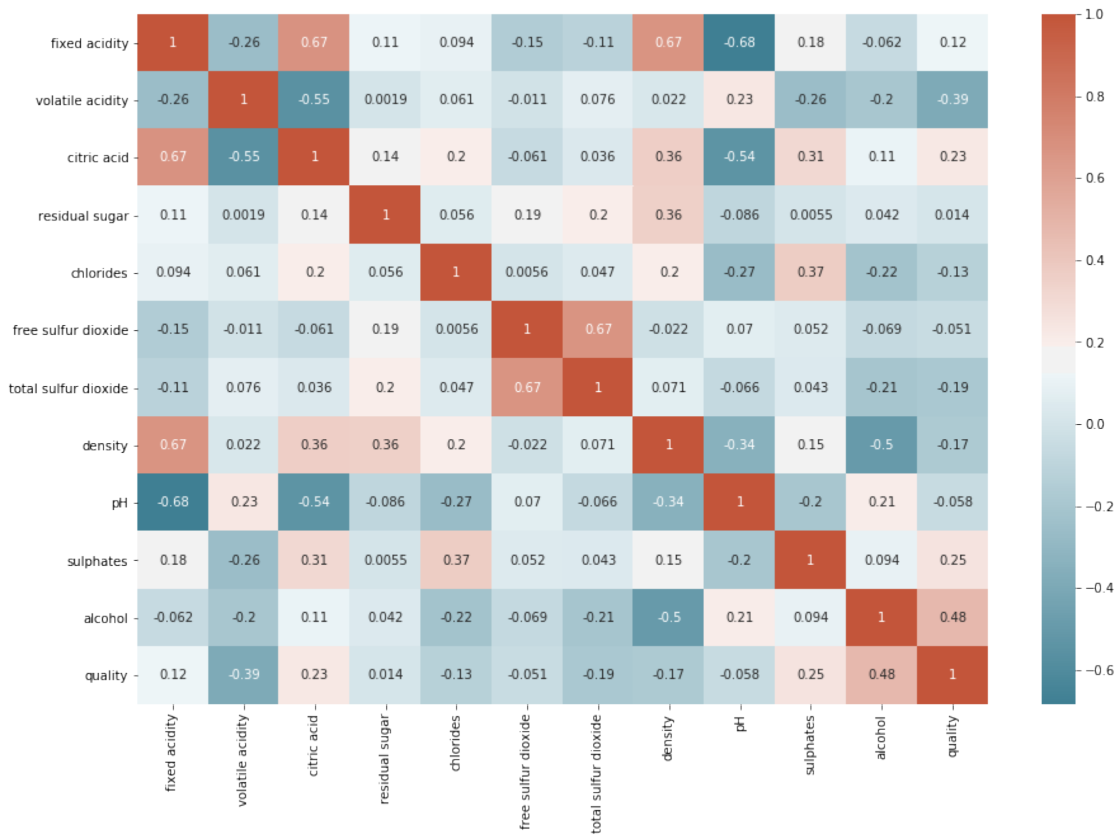
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
```

```
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Since there are no null values, this is a user-friendly database!

[9]:
```
#Correlation matrix to check correlation between the variables we work with
corr=wine.corr()
plt.pyplot.subplots(figsize=(15,10))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,␣
 ↪annot=True, cmap=sns.diverging_palette(220,20,as_cmap=True))
```

[9]: <matplotlib.axes._subplots.AxesSubplot at 0xa5e2730>



From the above correlation matrix, we learn that quality is mostly correlated to alcohol content

3 Classification of Good and Bad wines

[10]:
```
#Classify wines into good and bad
wine['goodquality']=[1 if x>=7 else 0 for x in wine['quality']]

#Separate feature variables and target variable
X=wine.drop(['quality','goodquality'],axis=1)
y=wine['goodquality']
```

[11]:
```
#Check number of good and bad wines (0=bad, 1=good)
wine['goodquality'].value_counts()
```

[11]:
```
0    1382
1     217
Name: goodquality, dtype: int64
```

There are 217 wines of good quality and 1382 wines of bad quality

[12]:
```
wine.head()
```

[12]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality  goodquality
0      9.4        5            0
1      9.8        5            0
2      9.8        5            0
3      9.8        6            0
4      9.4        5            0
```

4 Split Data into Training and Testing Set

[13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size= 0.2,␣
 ↪random_state= 0)
```

5 Different Machine Learning Models

We look at which model is the most accurate in classify good vs bad wine

This is done by obtaining

1. Accuracy = (True Positives + True Negatives)/All Samples
2. Precision = True Positives/(True Positives + False Positives)
3. Recall = True Positives/(True Positives + False Negatives)
4. Area under ROC = Quite useful evaluation metric when class is imbalanced. Value lies between 0.5 - 1. 1 means perfect classifier. 0.5 means worthless classifier.

6 1. Decision Tree

[14]:
```
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
model1=DecisionTreeClassifier(random_state=1)
model1.fit(X_train, y_train)
y_pred1=model1.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred1)
print(cnf_matrix)
print(classification_report(y_test, y_pred1))
```

```
[[266  24]
 [  7  23]]
              precision    recall  f1-score   support

           0       0.97      0.92      0.94       290
           1       0.49      0.77      0.60        30

    accuracy                           0.90       320
   macro avg       0.73      0.84      0.77       320
weighted avg       0.93      0.90      0.91       320
```

[15]:
```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred1))
print("Precision:",metrics.precision_score(y_test,y_pred1, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred1, average='weighted'))
y_pred_prob = model1.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.903125
Precision: 0.9288904800872885
Recall: 0.903125
Area under ROC curve score : 0.8419540229885057
```

7 2. Logistic Regression

[16]:
```
from sklearn.linear_model import LogisticRegression
model2 = LogisticRegression()
model2.fit(X_train,y_train)
y_pred2 = model2.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred2)
print(cnf_matrix)
print(classification_report(y_test, y_pred2))
```

```
[[281   9]
 [ 18  12]]
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       290
           1       0.57      0.40      0.47        30

    accuracy                           0.92       320
   macro avg       0.76      0.68      0.71       320
weighted avg       0.91      0.92      0.91       320


C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

[17]:
```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred2))
print("Precision:",metrics.precision_score(y_test,y_pred2, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred2, average='weighted'))
y_pred_prob = model2.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.915625
Precision: 0.9052645723841378
Recall: 0.915625
Area under ROC curve score : 0.8717241379310345
```

8 3. Random Forest

[18]:
```
from sklearn.ensemble import RandomForestClassifier
model3 = RandomForestClassifier(n_estimators = 200)
model3.fit(X_train,y_train)
y_pred3 = model3.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred3)
print(cnf_matrix)
print(classification_report(y_test, y_pred3))
```

```
[[280  10]
```

```
 [ 13  17]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       290
           1       0.63      0.57      0.60        30

    accuracy                           0.93       320
   macro avg       0.79      0.77      0.78       320
weighted avg       0.93      0.93      0.93       320
```

[19]:
```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred3))
print("Precision:",metrics.precision_score(y_test,y_pred3, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred3, average='weighted'))
y_pred_prob = model3.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.928125
Precision: 0.9250687334091772
Recall: 0.928125
Area under ROC curve score : 0.9337931034482759
```

9 4. Support Vector Machine

[20]:
```
from sklearn.svm import SVC
model4 = SVC(probability=True)
model4.fit(X_train, y_train)
y_pred4 = model4.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred4)
print(cnf_matrix)
print(metrics.classification_report(y_test,y_pred4))
```

```
C:\Users\arai2\anaconda3\Anaconda\lib\site-packages\sklearn\svm\base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

```
[[284   6]
 [ 22   8]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       290
           1       0.57      0.27      0.36        30

    accuracy                           0.91       320
   macro avg       0.75      0.62      0.66       320
weighted avg       0.89      0.91      0.90       320
```

[21]:
```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred4))
print("Precision:",metrics.precision_score(y_test,y_pred4, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred4, average='weighted'))
y_pred_prob = model4.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.9125
Precision: 0.8946661998132587
Recall: 0.9125
Area under ROC curve score : 0.8275862068965517
```

10 5. K-Neighbors Classifier

[22]:
```
from sklearn.neighbors import KNeighborsClassifier
model5=KNeighborsClassifier()
model5.fit(X_train,y_train)
y_pred5=model5.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred5)
print(cnf_matrix)
print(metrics.classification_report(y_test,y_pred5))
```

```
[[278  12]
 [ 18  12]]
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       290
           1       0.50      0.40      0.44        30

    accuracy                           0.91       320
   macro avg       0.72      0.68      0.70       320
weighted avg       0.90      0.91      0.90       320
```

[23]:
```
from sklearn import metrics

print("Accuracy:",metrics.accuracy_score(y_test,y_pred5))
print("Precision:",metrics.precision_score(y_test,y_pred5, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred5, average='weighted'))
y_pred_prob =model5.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.90625
Precision: 0.8980152027027026
Recall: 0.90625
Area under ROC curve score : 0.8043103448275861
```

11 6. Stochastic Gradient Descent Classifier

[24]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
sgd = SGDClassifier(penalty=None)
sg=sgd.fit(X_train, y_train)
calibrator=CalibratedClassifierCV(sg, cv='prefit')
model6=calibrator.fit(X_train,y_train)
y_pred6 = model6.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred6)
print(cnf_matrix)
print(metrics.classification_report(y_test,y_pred6))
```

```
[[279  11]
 [ 24   6]]
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       290
           1       0.35      0.20      0.26        30

    accuracy                           0.89       320
   macro avg       0.64      0.58      0.60       320
weighted avg       0.87      0.89      0.88       320


C:\Users\arai2\anaconda3\Anaconda\lib\site-packages\sklearn\calibration.py:453:
RuntimeWarning: overflow encountered in exp
  E = np.exp(AB[0] * F + AB[1])
C:\Users\arai2\anaconda3\Anaconda\lib\site-packages\sklearn\calibration.py:455:
RuntimeWarning: invalid value encountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
```

[25]:
```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred6))
print("Precision:",metrics.precision_score(y_test,y_pred6, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred6, average='weighted'))
y_pred_prob = model6.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.890625
Precision: 0.8675560570762958
Recall: 0.890625
Area under ROC curve score : 0.8287356321839082
```

12 7. AdaBoost

[26]:
```
from sklearn.ensemble import AdaBoostClassifier
model7 = AdaBoostClassifier(random_state=1)
model7.fit(X_train, y_train)
```

```
y_pred7 = model7.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred7)
print(cnf_matrix)
print(classification_report(y_test, y_pred7))
```

```
[[278  12]
 [ 17  13]]
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       290
           1       0.52      0.43      0.47        30

    accuracy                           0.91       320
   macro avg       0.73      0.70      0.71       320
weighted avg       0.90      0.91      0.91       320
```

[27]:
```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred7))
print("Precision:",metrics.precision_score(y_test,y_pred7, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred7, average='weighted'))
y_pred_prob = model7.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.909375
Precision: 0.9027754237288136
Recall: 0.909375
Area under ROC curve score : 0.8504597701149424
```

13 8. Gradient Boosting

[28]:
```
from sklearn.ensemble import GradientBoostingClassifier
model8 = GradientBoostingClassifier(random_state=1)
model8.fit(X_train, y_train)
y_pred8 = model8.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test,y_pred8)
print(cnf_matrix)
print(classification_report(y_test, y_pred8))
```

```
[[276  14]
 [ 13  17]]
              precision    recall  f1-score   support

           0       0.96      0.95      0.95       290
           1       0.55      0.57      0.56        30

    accuracy                           0.92       320
   macro avg       0.75      0.76      0.76       320
weighted avg       0.92      0.92      0.92       320
```

[29]:
```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred8))
print("Precision:",metrics.precision_score(y_test,y_pred8, average='weighted'))
print("Recall:",metrics.recall_score(y_test,y_pred8, average='weighted'))
y_pred_prob = model8.predict_proba(X_test)[::,1]
print("Area under ROC curve score :",metrics.roc_auc_score(y_test,y_pred_prob))
```

```
Accuracy: 0.915625
Precision: 0.9168957193883246
Recall: 0.915625
Area under ROC curve score : 0.9048275862068965
```

14 Individual Accuracies

[30]:
```
acDT = round(metrics.accuracy_score(y_test, y_pred1),2)

acLR = round(metrics.accuracy_score(y_test, y_pred2),2)

acRF = round(metrics.accuracy_score(y_test, y_pred3),2)

acSVM = round(metrics.accuracy_score(y_test, y_pred4),2)

acKNN = round(metrics.accuracy_score(y_test, y_pred5),2)

acSGD = round(metrics.accuracy_score(y_test, y_pred6),2)

acAB = round(metrics.accuracy_score(y_test, y_pred7),2)

acGB = round(metrics.accuracy_score(y_test, y_pred8),2)

list_accuracy_score  = [acDT, acLR, acRF, acSVM, acKNN, acSGD, acAB, acGB]
list_accuracy_score
```

[30]:
```
[0.9, 0.92, 0.93, 0.91, 0.91, 0.89, 0.91, 0.92]
```

15 Individual Precisions

[31]:
```
prDT = round(metrics.precision_score(y_test, y_pred1, average='weighted'),2)

prLR = round(metrics.precision_score(y_test, y_pred2, average='weighted'),2)

prRF = round(metrics.precision_score(y_test, y_pred3, average='weighted'),2)

prSVM = round(metrics.precision_score(y_test, y_pred4, average='weighted'),2)

prKNN = round(metrics.precision_score(y_test, y_pred5, average='weighted'),2)
```

```
prSGD = round(metrics.precision_score(y_test, y_pred6, average='weighted'),2)

prAB = round(metrics.precision_score(y_test, y_pred7, average='weighted'),2)

prGB = round(metrics.precision_score(y_test, y_pred8, average='weighted'),2)

list_precision_score  = [prDT, prLR, prRF, prSVM, prKNN, prSGD, prAB, prGB]
list_precision_score
```

[31]:
```
[0.93, 0.91, 0.93, 0.89, 0.9, 0.87, 0.9, 0.92]
```

16 Individual Recalls

[32]:
```
reDT = round(metrics.recall_score(y_test, y_pred1, average='weighted'),2)

reLR = round(metrics.recall_score(y_test, y_pred2, average='weighted'),2)

reRF = round(metrics.recall_score(y_test, y_pred3, average='weighted'),2)

reSVM = round(metrics.recall_score(y_test, y_pred4, average='weighted'),2)

reKNN = round(metrics.recall_score(y_test, y_pred5, average='weighted'),2)

reSGD = round(metrics.recall_score(y_test, y_pred6, average='weighted'),2)

reAB = round(metrics.recall_score(y_test, y_pred7, average='weighted'),2)

reGB = round(metrics.recall_score(y_test, y_pred8, average='weighted'),2)

list_recall_score  = [reDT, reLR, reRF, reSVM, reKNN, reSGD, reAB, reGB]
list_recall_score
```

[32]:
```
[0.9, 0.92, 0.93, 0.91, 0.91, 0.89, 0.91, 0.92]
```

17 Individual Area under ROC Scores

[33]:
```
rocDT = round(metrics.roc_auc_score(y_test,  model1.predict_proba(X_test)[::
 ↪,1]),2)

rocLR = round(metrics.roc_auc_score(y_test,  model2.predict_proba(X_test)[::
 ↪,1]),2)

rocRF = round(metrics.roc_auc_score(y_test,  model3.predict_proba(X_test)[::
 ↪,1]),2)
```

```
rocSVM = round(metrics.roc_auc_score(y_test,  model4.predict_proba(X_test)[::
 ↪,1]),2)

rocKNN = round(metrics.roc_auc_score(y_test,  model5.predict_proba(X_test)[::
 ↪,1]),2)

rocSGD = round(metrics.roc_auc_score(y_test,  model6.predict_proba(X_test)[::
 ↪,1]),2)

rocAB = round(metrics.roc_auc_score(y_test,  model7.predict_proba(X_test)[::
 ↪,1]),2)

rocGB = round(metrics.roc_auc_score(y_test,  model8.predict_proba(X_test)[::
 ↪,1]),2)

list_roc_auc_score  = [rocDT, rocLR, rocRF, rocSVM, rocKNN, rocSGD, rocAB,␣
 ↪rocGB]
list_roc_auc_score
```

[33]:
```
[0.84, 0.87, 0.93, 0.83, 0.8, 0.83, 0.85, 0.9]
```

18 Accuracy, Precision, Recall and Area under ROC score together

[34]:
```
df = pd.DataFrame(
    list(zip(
        ['DT', 'LR', 'RF', 'SVM', 'KNN', 'SGD', 'AB',␣
 ↪'GB'],list_accuracy_score, list_precision_score, list_recall_score,␣
 ↪list_roc_auc_score)),
    columns = ['Model', 'Accuracy','Precision','Recall','Area under ROC curve␣
 ↪score']
)

df = pd.DataFrame.from_dict(df)

print(df)
```

```
  Model  Accuracy  Precision  Recall  Area under ROC curve score
0    DT      0.90       0.93    0.90                        0.84
1    LR      0.92       0.91    0.92                        0.87
2    RF      0.93       0.93    0.93                        0.93
3   SVM      0.91       0.89    0.91                        0.83
4   KNN      0.91       0.90    0.91                        0.80
5   SGD      0.89       0.87    0.89                        0.83
6    AB      0.91       0.90    0.91                        0.85
7    GB      0.92       0.92    0.92                        0.90
```

From the above table, we can assure that Random Forest is the most appropriate classifier for the given dataset since it has the highest accuracy, precision, recall and area under ROC curve score

19 Finding the best Model Evaluation Using Jaccard Index, F1-Score and Log Loss

[35]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

20 Jaccard Similarity

[36]:
```
jcDT = round(jaccard_similarity_score(y_test, y_pred1),2)

jcLR = round(jaccard_similarity_score(y_test, y_pred2),2)

jcRF = round(jaccard_similarity_score(y_test, y_pred3),2)

jcSVM = round(jaccard_similarity_score(y_test, y_pred4),2)

jcKNN = round(jaccard_similarity_score(y_test, y_pred5),2)

jcSGD = round(jaccard_similarity_score(y_test, y_pred6),2)

jcAB = round(jaccard_similarity_score(y_test, y_pred7),2)

jcGB = round(jaccard_similarity_score(y_test, y_pred8),2)

list_jaccard_similarity  = [jcDT, jcLR, jcRF, jcSVM, jcKNN, jcSGD, jcAB, jcGB]
list_jaccard_similarity
```

```
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
```

```
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
C:\Users\arai2\anaconda3\Anaconda\lib\site-
packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It
will be removed in version 0.23. This implementation has surprising behavior for
binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

[36]:
```
[0.9, 0.92, 0.93, 0.91, 0.91, 0.89, 0.91, 0.92]
```

21 F1_score

[37]:
```
fDT = round(f1_score(y_test, y_pred1, average='weighted'), 2)
fLR = round(f1_score(y_test, y_pred2, average='weighted'), 2)
fRF = round(f1_score(y_test, y_pred3, average='weighted'), 2)
fSVM = round(f1_score(y_test, y_pred4, average='weighted'),2 )
fKNN = round(f1_score(y_test, y_pred5, average='weighted'), 2)
fSGD = round(f1_score(y_test, y_pred6, average='weighted'),2 )
fAB = round(f1_score(y_test, y_pred7, average='weighted'),2 )
fGB = round(f1_score(y_test, y_pred8, average='weighted'),2 )
```

```
list_f1_score = [fDT, fLR, fRF, fSVM, fKNN, fSGD, fAB, fGB]
list_f1_score
```

[37]: [0.91, 0.91, 0.93, 0.9, 0.9, 0.88, 0.91, 0.92]

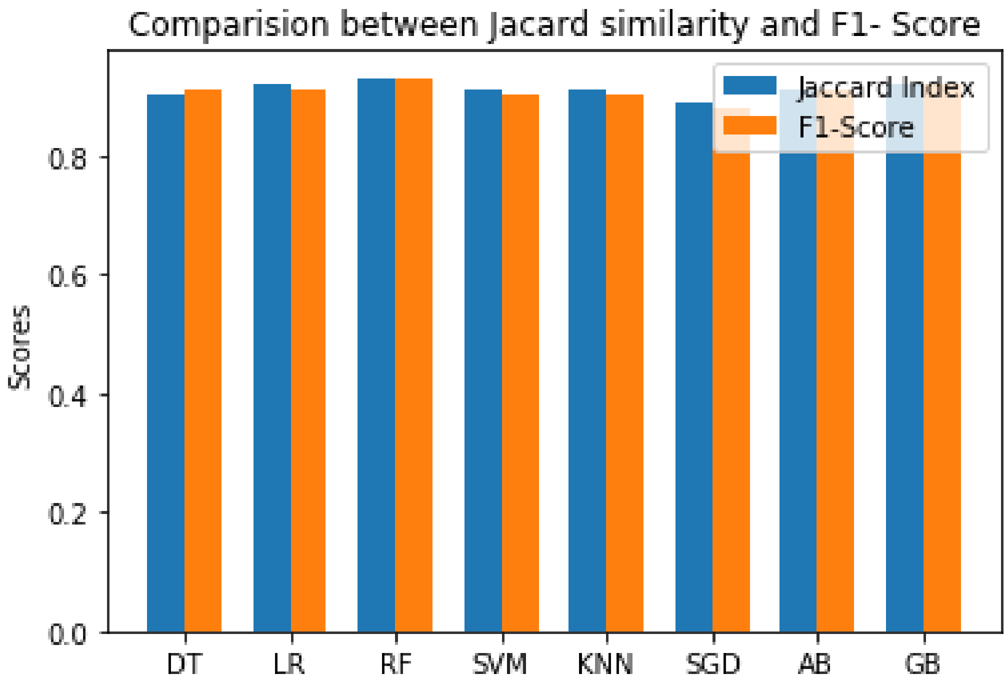
22 Log Loss

[38]:
```
lgloss_DT = round(log_loss(y_test, model1.predict_proba(X_test)), 2)
lgloss_LR = round(log_loss(y_test, model2.predict_proba(X_test)), 2)
lgloss_RF =round(log_loss(y_test, model3.predict_proba(X_test)), 2)
lgloss_SVM =round(log_loss(y_test, model4.predict_proba(X_test)), 2)
lgloss_KNN =round(log_loss(y_test, model5.predict_proba(X_test)), 2)
lgloss_SGD =round(log_loss(y_test, model6.predict_proba(X_test)), 2)
lgloss_AB =round(log_loss(y_test, model7.predict_proba(X_test)), 2)
lgloss_GB =round(log_loss(y_test, model8.predict_proba(X_test)), 2)

list_log_loss = [lgloss_DT, lgloss_LR, lgloss_RF, lgloss_SVM, lgloss_KNN,␣
 ↪lgloss_SGD, lgloss_AB, lgloss_GB]
list_log_loss
```

[38]: [3.35, 0.24, 0.19, 0.25, 1.02, 0.27, 0.65, 0.22]

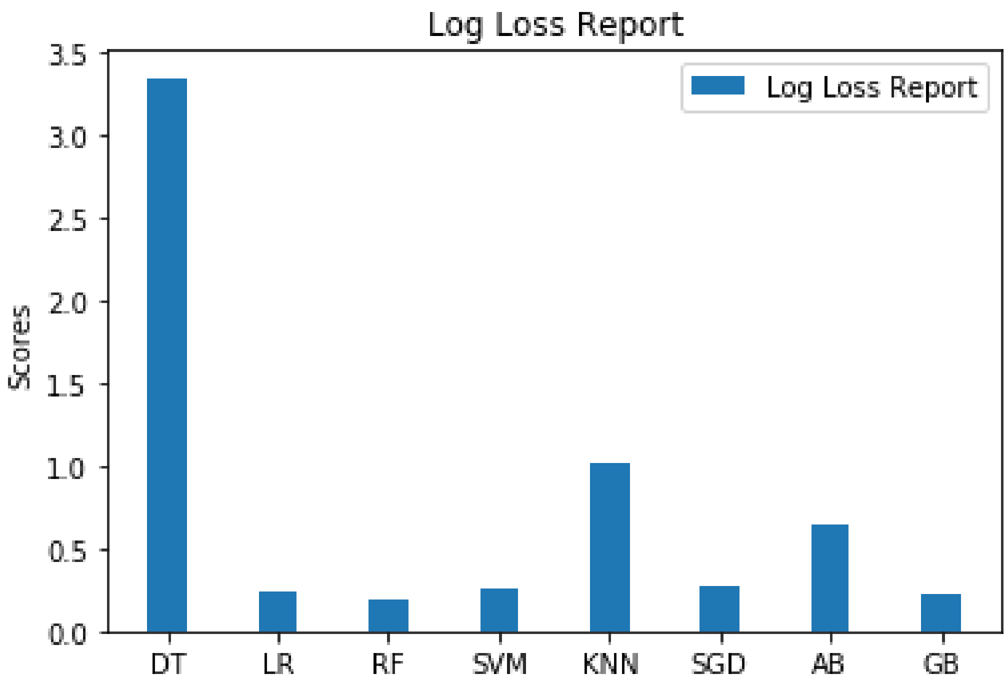
23 Final Report on Evaluation

[39]:
```
ind = np.arange(8)
width = 0.35
plot.bar(ind, list_jaccard_similarity, width, label='Jaccard Index')
plot.bar(ind + width,  list_f1_score, width,
    label='F1-Score')

plot.ylabel('Scores')
plot.title('Comparision between Jacard similarity and F1- Score')

plot.xticks(ind + width / 2, ('DT', 'LR', 'RF', 'SVM', 'KNN', 'SGD', 'AB',␣
 ↪'GB'))
plot.legend(loc='best')
plot.show()
```



[40]:

```
width = 0.35
plot.bar(ind, list_log_loss, width, label='Log Loss Report')

plot.ylabel('Scores')
plot.title('Log Loss Report')

plot.xticks(ind , ('DT', 'LR', 'RF', 'SVM', 'KNN', 'SGD', 'AB', 'GB'))
plot.legend(loc='best')
plot.show()
```



24 F1-score, Jaccard Similarity and Log Loss together:

[41]:
```
df = pd.DataFrame(
    list(zip(
        ['DT', 'LR', 'RF', 'SVM', 'KNN', 'SGD', 'AB',␣
 ↪'GB'],list_jaccard_similarity, list_f1_score, list_log_loss)),
    columns = ['Model', 'Jaccard','F1-Score','Log-Loss']
)

df = pd.DataFrame.from_dict(df)

print(df)
```

```
  Model  Jaccard  F1-Score  Log-Loss
0    DT     0.90      0.91      3.35
1    LR     0.92      0.91      0.24
2    RF     0.93      0.93      0.19
3   SVM     0.91      0.90      0.25
4   KNN     0.91      0.90      1.02
5   SGD     0.89      0.88      0.27
6    AB     0.91      0.91      0.65
7    GB     0.92      0.92      0.22
```

Highest Jaccard Similarity = Random Forest

Highest F1-score = Random Forest

Lowest Log Loss = Random Forest Therefore, Random Forest is the most ideal classifier for this particular dataset.

[42]:
```
X_predict = list(model3.predict(X_test))
predicted_df = {'predicted_values': X_predict, 'original_values': y_test}
#creating new dataframe
pd.DataFrame(predicted_df).head(20)
```

[42]:

	predicted_values	original_values
1109	0	0
1032	0	0
1002	1	1
487	0	0
979	0	0
1054	0	0
542	0	0
853	0	0
1189	0	0
412	0	0
1099	0	0
475	0	0
799	0	0
553	0	0
1537	0	0
1586	0	0
805	1	1
1095	0	0
1547	0	0
18	0	0

[43]:
```
pd.DataFrame(predicted_df)['predicted_values'].value_counts()
```

[43]:
```
0    293
1     27
Name: predicted_values, dtype: int64
```

[44]:
```
pd.DataFrame(predicted_df)['original_values'].value_counts()
```

[44]:
```
0    290
1     30
Name: original_values, dtype: int64
```

The above reading confirms the accuracy of the Random Forest Classifier

[ ]:
```
```