LAB CYCLE 4

Experiment No:4

Date:10/02/22

Aim:

Convert all the values in quality attribute to 0 (bad) if the value is lessthan '5', to 1 (good) if the value is '5' or '6' and to 2 (great) otherwise.Normalize all the other attributes byZ-score normalization, and segregate them into 4 equal spaced bins each giving thevalues between [0 to 3], and replace the values for that attributewith the number corresponding to the interval they belong.

For example, suppose after normalization an attribute has valuesbetween [-0.5,1.5], i.e., minimum value of the attribute is -0.5 andmaximum value is 1.5, then form 4 bins:

bin 0: [-0.5,0.0],
bin 1: [0.0,0.5],
bin 2: [0.5,1.0],
bin 3: [1.0,1.5].

For example, if a data instance has a value of 0.73 for that attribute, replace 0.73 with 2. Use this dataset for constructing a Decision Tree.

Problem Statement

1. **Implement Decision tree algorithm using information gain to choose which attribute to split at each point. Stop splitting a node if it has less than 10 data points. Do NOT use scikit-learn for this part.**

2. **Test out the implementation of Decision Tree Classifier from scikit-learn package, Using information gain. Here also stop splitting a node if it hasless than 10 data points.**

3. **Cross validate the classifiers with 3-folds and print the mean macro accuracy, macro precision and macro recall for both the classifiers. You may or may not use the scikit-learn implementations for computing these metrics and cross validation.**

Source Code:

In [1]:
```
import numpy as np
import pandas as pd
from pprint import pprint
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
dt=pd.read_csv("winequality-red.csv")
```

In [3]:
```
dt
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

In [4]:
```
n=dt.shape
n
```

Out[4]:
```
(1599, 12)
```

In [51]:
```
dt.head(8)
```

Out[51]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7

In [52]:
```
dt.loc[dt["quality"]<5,"quality"]=0
dt.loc[dt["quality"]==5,"quality"]=1
dt.loc[dt["quality"]==6,"quality"]=1
dt.loc[dt["quality"]<6,"quality"]=2
```

In [53]:
```
dt.head(8)
```

Out[53]:

	fixed acidity	volatile acidity	citric acid	residual c sugar	hlorides	free sulfur dioxide	total sulfur dioxide	density	pH	su phates	a cohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	2
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	2
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	2
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	2
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	2
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	2
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	2
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7

In [54]:

```
def normalize(x):
    xnew=((x-np.mean(x))/np.std(x))

#print(xnew)
    return xnew
```

In [55]:

```
dt.iloc[:,0:11]=dt.iloc[:,0:11].apply(normalize)
```

In [56]:

```
dt.head(8)
```

Out[56]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	-0.528360	0.961877	-1.391472	-0.453218	-0.243707	-0.466193	-0.379133	0.558274	1.288643	-0.579207	-0.960246	2
1	-0.298547	1.967442	-1.391472	0.043416	0.223875	0.872638	0.624363	0.028261	-0.719933	0.128950	-0.584777	2
2	-0.298547	1.297065	-1.186070	-0.169427	0.096353	-0.083669	0.229047	0.134264	-0.331177	-0.048089	-0.584777	2
3	1.654856	-1.384443	1.484154	-0.453218	-0.264960	0.107592	0.411500	0.664277	-0.979104	-0.461180	-0.584777	2
4	-0.528360	0.961877	-1.391472	-0.453218	-0.243707	-0.466193	-0.379133	0.558274	1.288643	-0.579207	-0.960246	2
5	-0.528360	0.738418	-1.391472	-0.524166	-0.264960	-0.274931	-0.196679	0.558274	1.288643	-0.579207	-0.960246	2
6	-0.241094	0.403229	-1.083370	-0.666062	-0.392483	-0.083669	0.381091	-0.183745	-0.072005	-1.169337	-0.960246	2
7	-0.585813	0.682553	-1.391472	-0.949853	-0.477498	-0.083669	-0.774449	-1.137769	0.511130	-1.110324	-0.397043	7

In [57]:

```
bin_labels_4=[0,1,2,3]
#dt['fixed acidity'] = pd.qcut(dt['fixed acidity'],q=4,label=bin_labels_4)
for i in range(0,11):
    dt.iloc[:,i]=pd.qcut(dt.iloc[:,i],q=4,labels=bin_labels_4)
dt.head(8)
```

Out[57]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	1	3	0	0	1	1	1	2	3	1	0	2
1	1	3	0	2	3	3	3	2	0	2	1	2
2	1	3	0	2	3	2	2	2	1	2	1	2
3	3	0	3	0	1	2	2	3	0	1	1	2
4	1	3	0	0	1	1	1	2	3	1	0	2
5	1	3	0	0	1	1	2	2	3	1	0	2
6	1	2	0	0	0	2	2	1	1	0	0	2
7	1	3	0	0	0	2	0	0	2	0	1	7

In [58]:

```
traincount=int(dt.shape[0]*0.8)
traincount
```

Out[58]: 1279

In [59]:
```
def train_test_split(dt):
    training_data=dt.iloc[:traincount].reset_index(drop=True)
    testing_data=dt.iloc[traincount:].reset_index(drop=True)
    return training_data,testing_data
training_data=train_test_split(dt)[0]
testing_data=train_test_split(dt)[1]
```

In [60]:
```
training_data.shape
```

Out[60]: (1279, 12)

In [61]:
```
testing_data.shape
```

Out[61]: (320, 12)

In [62]:
```
#Compute entropy
```

In [63]:
```
def entropy(class_label):
    values,counts=np.unique(class_label,return_counts=True)

    for i in range(len(values)):
        entropy=np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts))])
    return entropy
```

In [64]:
```
#Info Gain
```

In [65]:
```
def InfoGain(data,split_attribute_name,class_label="equality"):
    total_entropy=entropy(data[class_label])
    vals,counts=np.unique(data[split_attribute_name],return_counts=True)
    #Calculate the weighted entropy
    for i in range(len(vals)):
        Weighted_Entropy=np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==va

    #formula for information gain
    Information_Gain=total_entropy-Weighted_Entropy
    return Information_Gain
```

In [66]:
```
def   ID3(data,originaldata,features,class_label="quality",parent_node_class=None):
    #if all class_label values are same, return that value
    if len(np.unique(data[class_label]))<=1:
        return np.unique(data[class_label])[0]

    #if the dataset is empty or below some threshold value,terminate recursion
    elif len(data)==0:
        #find the counts of distinct values of class_label, then find the maximum count of them--> majority c
        return    np.unique(originaldata[class_label])[np.argmax(np.unique(originaldata[class_label],return_coun

    #if the feature space is empty,terminate recursion
    elif len(features)==0:
        return parent_node_class

    #If none of the above condition holds true form the subtrees

    else:
        #Find the counts of distinct values of class_label, then find the maximum count of them-->majority cl
        parent_node_class=np.unique(data[class_label])[np.argmax(np.unique(data[class_label],return_counts=Tr

    #Select the feature which best splits the dataset, feature having maximum information gain

    for feature in features:
        item_values=[InfoGain(data,feature,class_label)] #Return the infogain values
    best_feature_index=np.argmax(item_values)
    best_feature=features[best_feature_index]

    #Create the tree structure as a nested dictionary
```

```
    tree={best_feature:{}}

    #Remove the feature with the best info gain
    features=[i for i in features if i!=best_feature]

    #Form subtrees down the root node by calling ID3 recursively

    for value in np.unique(data[best_feature]):
        value=value
        sub_data=data.where(data[best_feature]==value).dropna()
        #call the ID3 algorthm
        subtree=ID3(sub_data,dt,features,class_label,parent_node_class)
        #Add the subtree
        tree[best_feature][value]=subtree
    return(tree)
```

In [67]:

```
tree = ID3(training_data,training_data,training_data.columns[:-1])
pprint(tree)
```

```
{'fixed acidity': {0: {'volatile acidity': {0: {'citric acid': {0: 2.0,
                                                                1: 2.0,
                                                                2: {'residual sugar': {0: {'chlorides': {0:
{'free sulfur dioxide': {0: 7.0,

2: 2.0,

3: 2.0}},
                                                                                                    1:
2.0,
                                                                                                    2:
2.0}},
                                                                                    1: 2.0,
                                                                                    2: {'chlorides': {0:
7.0,
                                                                                                    3:
2.0}},
                                                                                    3: 2.0}},
                                                                3: {'residual sugar': {0: 2.0,
                                                                                       1: 7.0,
                                                                                       2: 2.0}}}},
                                            1: {'citric acid': {0: {'residual sugar': {0: {'chlorides': {0:
{'free sulfur dioxide': {2: {'total sulfur dioxide': {1: 2.0,

2: 2.0,

3: 7.0}},

3: {'total sulfur dioxide': {1: 2.0,

3: 8.0}}}},
                                                                                                    1:
2.0,
                                                                                                    2:
2.0}},
                                                                                    1: {'chlorides': {0:
{'free sulfur dioxide': {0: 2.0,

1: 2.0,

2: 7.0}},
                                                                                                    1:
2.0,
                                                                                                    3:
2.0}},
                                                                                    2: {'chlorides': {0:
{'free sulfur dioxide': {1: 7.0,

2: 2.0}},
                                                                                                    1:
2.0}}}},
                                                                1: {'residual sugar': {0: 2.0,
                                                                                       1: {'chlorides': {0:
{'free sulfur dioxide': {0: 2.0,
```

```
1: 2.0,
2: {'chlorides': {0:
 2.0,
1:
 2.0,
2:
 2.0,
3:
 {'free sulfur dioxide': {1: 2.0,
3: 7.0}}}},
3: {'chlorides': {1:
 2.0,
2:
 {'free sulfur dioxide': {0: {'total sulfur dioxide': {0: 8.0,
1: 2.0,
2: 7.0}},
1: 2.0}},
3:
 {'free sulfur dioxide': {0: {'total sulfur dioxide': {0: {'density': {2: 7.0,
3: {'pH': {0: 7.0,
1: 2.0}}}}}},
1: 2.0,

3: 2.0}}}}}}}},
3: {'citric acid': {1: 2.0,
2: {'residual sugar': {0: 2.0,
1: 2.0,
2: 2.0,
3: {'chlorides': {1:
 2.0,
2:
 7.0,
3:
 2.0}}}},
3: {'residual sugar': {0: 2.0,
2: 2.0,
3: {'chlorides': {1:
 2.0,
2:
 2.0,
3:
 {'free sulfur dioxide': {0: 7.0,
1: 2.0}}}}}}}}}}}}
```

In [68]:

```
def predict(query,tree,default = 1):
    #1.
    for key in list(query.keys()):
        if key in list(tree.keys()):
            #2.
            try:
                result = tree[key][query[key]]
            except:
                return default

            #3.
            result = tree[key][query[key]]
            #4.
            if isinstance(result,dict):
                return predict(query,result)

            else:
                return result
```

In [69]:

```
def test(data,tree):
    #Create new query instances by simply removing the target feature column from the original dataset and
    #convert it to a dictionary
    queries = data.iloc[:,:-1].to_dict(orient = "records")

    #Create a empty DataFrame in whose columns the prediction of the tree are stored
```

```
    predicted = pd.DataFrame(columns=["predicted"])

    #Calculate the prediction accuracy
    for i in range(len(data)):
        predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)
    print('The prediction accuracy is: ',(np.sum(predicted["predicted"] == data["quality"])/len(data))*100,'%
```

In [70]:
```
test(testing_data,tree)
```

```
The prediction accuracy is:  78.75 %
```

In [ ]:
```

```

In [72]:
```
#Decision tree  implemenentation using Libraries

from sklearn.model_selection import train_test_split#/or decision rree object
from sklearn.tree import DecisionTreeClassifier#/or checking testing resutts
from sklearn.metrics import classification_report
```

In [73]
```
#to   divide  data  into  attributes  and labels,execute the following code:

X = dt.drop('quality', axis=1)
y = dt['quality']
```

In [74]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [75]
```
#TTraining  and Making  Predictions

classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

Out[75]:
```
DecisionTreeClassifier()
```

In [76]:
```
y_pred = classifier.predict(X_test)
```

In [77]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           2       0.93      0.92      0.92       282
           7       0.39      0.41      0.40        34
           8       0.00      0.00      0.00         4

    accuracy                           0.85       320
   macro avg       0.44      0.44      0.44       320
weighted avg       0.86      0.85      0.85       320
```

In [37]:
```
target = list(dt['quality'].unique())
feature_names = list(X.columns)
#We can aLso get a textuaL representation of the tree by using the export tree function from the SkLearn Libr
from sklearn.tree import export_text
r =export_text(classifier, feature_names=feature_names)
print(r)
```

```
|--- alcohol <= 2.50
|   |--- olatile acidity <= 0.50
|   |   |--- sulphates <= 1.50
|   |   |   |--- density <= 0.50
|   |   |   |   |--- pH <= 1.50
|   |   |   |   |   |--- sulphates <= 0.50
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- sulphates >  0.50
|   |   |   |   |   |   |--- class: 7
|   |   |   |   |--- pH >  1.50
|   |   |   |   |   |--- class: 2
|   |   |   |--- density >  0.50
|   |   |   |   |--- class: 2
|   |   |--- sulphates >  1.50
```

```
|   |   |   |   |   |   |   |   |   |   |   class: 2
|   |   |   |   |   |   |   |--- free sulfur dioxide >  0.50
|   |   |   |   |   |   |   |   |--- class: 7
|   |   |   |   |   |--- free sulfur dioxide >  1.50
|   |   |   |   |   |   |--- pH <= 0.50
|   |   |   |   |   |   |   |--- residual sugar <= 2.50
|   |   |   |   |   |   |   |   |--- class: 7
|   |   |   |   |   |   |   |--- residual sugar >  2.50
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- pH >  0.50
|   |   |   |   |   |   |   |--- class: 2
```

In [78]:

```
# K FOLD CROSS VALIDATION, K-=3

from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


X = dt.iloc[:,:-1]
y = dt.iloc[:, -1]

#Implementing cross validation

k=3
kf = KFold(n_splits=k, random_state=None)
model = LogisticRegression(solver= 'liblinear')

acc_score = []

for train_index , test_index  in kf.split(X):
    X_train,X_test=X.iloc[train_index,:],X.iloc[test_index,:]
    y_train , y_test = y[train_index] , y[test_index]

    model.fit(X_train,y_train)
    pred_values = model.predict(X_test)

    acc = accuracy_score(pred_values , y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/k

print('accuracy of each fold - {}'.format(acc_score))
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
accuracy of each fold - [0.874296435272045, 0.8667917448405253, 0.8780487804878049]
Avg accuracy : 0.8730456535334584
```