LAB CYCLE 4

EXPERIMENT NO:4

DATE:01/02/22

AIM: Data Set Preparation: Download the winequality-red.csv from this link . It contains various chemical properties of Red wine samples, along with the quality of wine. We want to train classifiers to predict the quality. We shall create two modified datasets from this data. Convert all the values in quality attribute to 0 (bad) if the value is less than '5', to 1 (good) if the value is '5' or '6' and to 2 (great) otherwise. Normalize all the other attributes by Z-score normalization, and segregate them into 4 equal spaced bins each giving the values between [0 to 3], and replace the values for that attribute with the number corresponding to the interval they belong.

For example, suppose after normalization an attribute has values between [-0.5,1.5], i.e., minimum value of the attribute is -0.5 and maximum 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 has less 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 WITH OUTPUT

In [22]:
```
#LAB CYCLE 4 - Decision Tree Classifier
```

In [5]:
```
import pandas as pd
import numpy as np
from pprint import pprint
```

In [6]:
```
df=pd.read_csv("winequality-red.csv")
```

In [7]:
```
#to know the no:of rows and columns
k=df.shape
k
```

Out[7]:
```
(1599, 12)
```

In [8]:
```
#to print first 10 rows
df.head(10)
```

Out[8]:

	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
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

In [9]:
```
#To preprocess the dataset. ie to change the values of classlabel 'quality' to a smaller range(0,1,2)
df.loc[df["quality"] < 5,"quality"]=0
df.loc[df["quality"] == 5,"quality"]=1
df.loc[df["quality"] == 6,"quality"]=1
df.loc[df["quality"] > 6,"quality"]=2
```

In [10]:
```
#to print first 10 rows again to show the changed values of class label
df.head(10)
```

Out[10]:

	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	1
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	1
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	1
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	1
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	1
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	1
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	2
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	2
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	1

In [11]:
```
#To preprocess remaining attribute values in Dataset
#we use 'Z-SCORE' normalization
#new value=[old value-mean(data)]/Standard deviation
#For that define a function named 'normalize'
def normalize(x):
    x_new=((x-np.mean(x))/np.std(x))
    return x_new
```

In [12]:
```
# We apply normalization from columns 0 to 11
df.iloc[:,0:11]=df.iloc[:,0:11].apply(normalize)
```

In [13]:
```
df.head(10)
```

Out[13]:

	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	1
1	-0.298547	1.967442	-1.391472	0.043416	0.223875	0.872638	0.624363	0.028261	-0.719933	0.128950	-0.584777	1
2	-0.298547	1.297065	-1.186070	-0.169427	0.096353	-0.083669	0.229047	0.134264	-0.331177	-0.048089	-0.584777	1
3	1.654856	-1.384443	1.484154	-0.453218	-0.264960	0.107592	0.411500	0.664277	-0.979104	-0.461180	-0.584777	1
4	-0.528360	0.961877	-1.391472	-0.453218	-0.243707	-0.466193	-0.379133	0.558274	1.288643	-0.579207	-0.960246	1
5	-0.528360	0.738418	-1.391472	-0.524166	-0.264960	-0.274931	-0.196679	0.558274	1.288643	-0.579207	-0.960246	1
6	-0.241094	0.403229	-1.083370	-0.666062	-0.392483	-0.083669	0.381091	-0.183745	-0.072005	-1.169337	-0.960246	1
7	-0.585813	0.682553	-1.391472	-0.949853	-0.477498	-0.083669	-0.774449	-1.137769	0.511130	-1.110324	-0.397043	2
8	-0.298547	0.291499	-1.288771	-0.382271	-0.307468	-0.657454	-0.865676	0.028261	0.316751	-0.520193	-0.866379	2
9	-0.470907	-0.155419	0.457144	2.526589	-0.349975	0.107592	1.688677	0.558274	0.251958	0.837107	0.072294	1

In [14]:
```
#to transform each column values, form 4 bins
#'qcut' function is used to form 4 bins
#each column values can be 0,1,2 or 3
#qcut(column_name,no:of bins,label)

bin_labels4=[0,1,2,3]
for i in range(0,11):
    df.iloc[:,i]=pd.qcut(df.iloc[:,i],q=4,labels=bin_labels4)
df.head(10)
```

Out[14]:

	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	1
1	1	3	0	2	3	3	3	2	0	2	1	1
2	1	3	0	2	3	2	2	2	1	2	1	1
3	3	0	3	0	1	2	2	3	0	1	1	1
4	1	3	0	0	1	1	1	2	3	1	0	1
5	1	3	0	0	1	1	2	2	3	1	0	1
6	1	2	0	0	0	2	2	1	1	0	0	1
7	1	3	0	0	0	2	0	0	2	0	1	2
8	1	2	0	1	1	1	0	2	2	1	0	2
9	1	1	2	3	1	2	3	2	2	3	2	1

In [15]:
```
#Spliting the data Set
#80% training dataset and 20% test dataset
traincount=int(df.shape[0]*0.8)
traincount
```

Out[15]:
```
1279
```

In [16]:
```
def train_test_split(df):
    training_data=df.iloc[:traincount].reset_index(drop=True)
    testing_data=df.iloc[traincount:].reset_index(drop=True)
    return training_data,testing_data
#for training_data index starts from 0
#for testing_data index starts from 1
training_data=train_test_split(df)[0]
testing_data=train_test_split(df)[1]
```

In [17]:
```
training_data.shape
```

Out[17]: (1279, 12)

In [18]:
```
testing_data.shape
```

Out[18]: (320, 12)

In [19]:
```
#Computing Entropy
def entropy(class_label_data):
    #'unique'method to find distint values in class_label_data & corresponding count of each values
    values,counts=np.unique(class_label_data,return_counts=True)
    for i in range(len(values)):
        #formula: -pi log2 pi for all i
        entropy=np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts))])
    return entropy
```

In [20]:
```
#Computing Information gain
def infogain(data,split_attribute_name,class_label="quality"):
    total_entropy=entropy(data[class_label])
    vals,counts=np.unique(data[split_attribute_name],return_counts=True)
    #calculating weighted entropy
    for i in range(len(vals)):
        weighted_entropy=np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==
                                                                               vals[i]).dropna()[class_label])])
    Information_gain=total_entropy-weighted_entropy
    return Information_gain
```

In [21]:
```
#implementing decision tree algorithm on training dataset
def ID3(data,orginaldata,features,class_label="quality",parent_node_class=None):
    #if all the class_label values are same,then return that value
    if len(np.unique(data[class_label])) <=1:
        return np.unique(data[class_label])[0]
    #if dataset is empty or below some threshold vale,then terminate recursion
    elif len(data)== 0:
         return np.unique(orginaldata[class_label])[np.argmax(np.unique(orginaldata[class_label],return_counts=T
        #if all feature space is empty,then terminate recursion
    elif len(features)== 0:
        return parent_node_class
    #if above conditions are not true,then form subtrees
    else:
        #find counts of distinct values of classlabel,then find max(count) from them ,select majority class labe
        parent_node_class=np.unique(data[class_label])[np.argmax(np.unique(data[class_label],return_counts=True)
    #select the features which best splits the dataset,feature having maximum Info Gain
    for feature in features:
        item_values=[infogain(data,feature,class_label)]
    best_feature_index=np.argmax(item_values)
    best_feature=features[best_feature_index]

    #create tree struct
    tree={best_feature:{}}

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

    #form subtrees by calling id3 function recursively

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

In [22]:
```
tree=ID3(training_data,training_data,training_data.columns[:-1])
pprint(tree)
```

```
{'free sulfur dioxide': {0: 0.0,

2: 1.0}}}},
                                                                                2: 1.0,
                                                                                3: 1.0}},
                                                                    2: {'residual sugar': {0: {'chlorides': {0:
{'free sulfur dioxide': {0: 2.0,

2: 1.0,

3: 1.0}},
                                                                                          1:
1.0,
                                                                                          2:
1.0}},
                                                                                1: 1.0,
                                                                                2: {'chlorides': {0:
2.0,
                                                                                          3:
1.0}},
```

In [23]:
```
def predict(query,tree,default = 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 [24]:
```
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 [25]:
```
test(testing_data,tree)
```

```
The prediction accuracy is:  83.75 %
```

In [37]:
```
#Decision tree implementation using libraries


from sklearn.model_selection import train_test_split#for decision tree object
from sklearn.tree import DecisionTreeClassifier#for checking testing results
from sklearn.metrics import classification_report
```

In [38]:
```
#to divide data into attributes and labels, execute the following code:

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

In [39]:
```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [40]:
```
#Training and Making Predictions

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

Out[40]:
```
DecisionTreeClassifier()
```

In [41]:
```
y_pred = classifier.predict(X_test)
```

In [42]:
```


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

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        14
           1       0.90      0.87      0.88       265
           2       0.55      0.68      0.61        41

    accuracy                           0.81       320
   macro avg       0.48      0.52      0.50       320
weighted avg       0.81      0.81      0.81       320
```

In [96]:
```
target = list(df['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 library
from sklearn.tree import export_text
r = export_text(classifier, feature_names=feature_names)
print(r)
```

```
|--- alcohol <= 2.50
|   |--- volatile acidity <= 0.50
|   |   |--- sulphates <= 2.50
|   |   |   |--- chlorides <= 0.50
|   |   |   |   |--- fixed acidity <= 0.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- fixed acidity >  0.50
|   |   |   |   |   |--- residual sugar <= 0.50
|   |   |   |   |   |   |--- free sulfur dioxide <= 1.50
|   |   |   |   |   |   |   |--- density <= 0.50
|   |   |   |   |   |   |   |   |--- sulphates <= 1.50
|   |   |   |   |   |   |   |   |   |--- pH <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |   |   |--- pH >  1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- sulphates >  1.50
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- density >  0.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- free sulfur dioxide >  1.50
```

In [ ]:

In [93]:
```
# 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 = df.iloc[:,:-1]
y = df.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 - [0.9210526315789473, 0.9736842105263158, 0.9470899470899471]
Avg accuracy : 0.9472755963984034
```

In [ ]:
```

```