DMBI-Assignment 2: Classification using Decision trees

Group No : 12

Abdul Azeez MS21W001

Gokul P MS21W018

Mohammed Zabi MS21W027

Senthilkumar MS21W037

Assignment 2: Classification using Decision trees

Problem statement:

Build decision tree classifiers as given in the tables below. Evaluate their prediction and validation as required.

Dataset

The Cars Evaluation data set consists of 7 attributes, 6 as feature attributes and 1 as the target attribute (car_evaluation). All the attributes are categorical. 1 buying vhigh, high, med, low 2 maintenance vhigh, high, med,low 3 doors 2, 3, 4, 5 , more 4 persons 2, 4, more 5 lug_boot small, med, big 6 safety low, med, high 7 car_evaluation unacc, acc, good, vgood

Q1: Build different decision tree models and evaluate performance using k-fold validation

CART

In [1]:
```
#IMPORT PYTHON MODULES

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
#This module needs to be separately invoked from tree library
from sklearn.model_selection import train_test_split,StratifiedKFold
#Stratified K-fold is a variation of KFold that returns stratified folds. The folds
from sklearn.metrics import confusion_matrix
#from sklearn.tree import export_graphviz
import pandas as pd
import numpy as np
#While pandas adopts many coding idioms from NumPy, the biggest difference is that p
import graphviz
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification #for bootstrapping
from sklearn.model_selection import KFold
#import category_encoders as ce
```

In [2]:
```
#READ AND EXPLORE DATA
#Use pandas read_csv function to upload Bikebuyer data from the computer folder (Mac
#Alternatively upload data file to jupyter homepage and use pd.read_csv("Bikebuyer.c
dtf=pd.read_csv("car_data.csv")
```

In [3]:
```
dtf['buying'],_ = pd.factorize(dtf['buying'])
dtf['maintenance'],_ = pd.factorize(dtf['maintenance'])
dtf['doors'],_ = pd.factorize(dtf['doors'])
dtf['persons'],_ = pd.factorize(dtf['persons'])
dtf['lug_boot'],_ = pd.factorize(dtf['lug_boot'])
dtf['safety'],_ = pd.factorize(dtf['safety'])
dtf
```

Out[3]:

	buying	maintenance	doors	persons	lug_boot	safety	car_evaluation
0	0	0	0	0	0	0	unacc
1	0	0	0	0	0	1	unacc
2	0	0	0	0	0	2	unacc
3	0	0	0	0	1	0	unacc
4	0	0	0	0	1	1	unacc
...	...	...	...	...	...	...	...
1723	3	3	3	2	1	1	good
1724	3	3	3	2	1	2	vgood
1725	3	3	3	2	2	0	unacc
1726	3	3	3	2	2	1	good
1727	3	3	3	2	2	2	vgood

1728 rows × 7 columns

In [4]:
```
#Extract relevant features from data
X=dtf[['buying','maintenance','doors','persons','lug_boot','safety']]
#'MaritalStatus' not included as scikit learn CART algorithm doesnt support categori
#type(X)
#np.shape(X) # dimension of dataframe
#X.dtypes
X.head()
```

Out[4]:

	buying	maintenance	doors	persons	lug_boot	safety
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	0	2
3	0	0	0	0	1	0
4	0	0	0	0	1	1

In [5]:
```
#DATA PREPARATION
#scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn
#Assign Bikebuyer as target variable y
y=dtf['car_evaluation']
#Split X into train and test data: https://scikit-learn.org/stable/modules/generated
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_sta
```

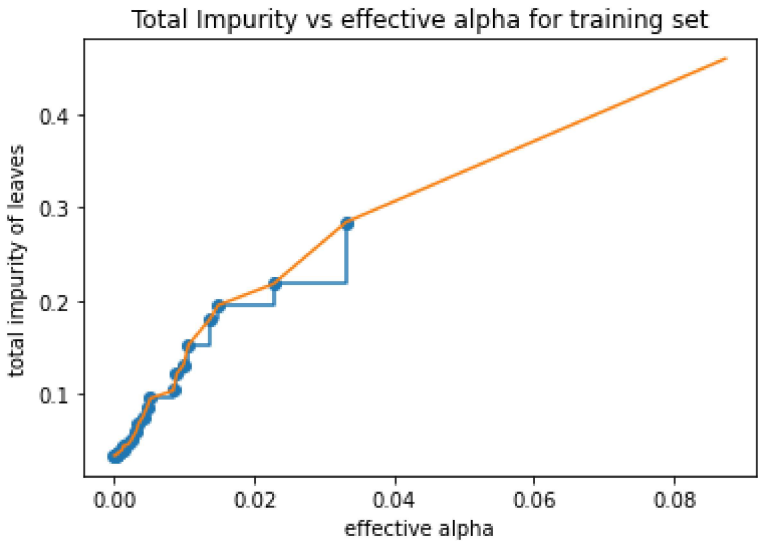
In [6]:
```
#MODEL BUILDING
```

```
#Specify decision tree model as dt using scikit learn "DecisionTreeClassifier" modul
dt=DecisionTreeClassifier(criterion='gini',
    min_samples_leaf=5 ,max_depth=10,
    ccp_alpha=0.0)
#Fit the model using fit() method
dt.fit(X_train, y_train)
```

Out[6]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [7]:
```
#TREE PRUNING USING CCP ALPHA
#Determining Cost Complexity Parameter (ccp_alpha) for post pruning the tree: https:
path = dt.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
#Using matplotlib.pyplot to plot the effect of varying ccp_alpha on error
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1],impurities[:-1],marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.plot(ccp_alphas, impurities)

dts = [] #tree building
for ccp_alpha in ccp_alphas:
    dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    dt.fit(X_train, y_train)
    dts.append(dt)
```



In [8]:
```
dts
```

Out[8]:
```
[DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                        max_depth=None, max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort='deprecated',
                        random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.00012004225487371664, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
```

```
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.00022224966045190832, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0004258890433780514, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0004390116749667314, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0005041774704696064, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0008544825960556308, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0009383874552413869, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0011853315224101737, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0012102168617898955, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0012964563526361278, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.002028572881183589, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.002466274722108776, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0031587376821157163, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
```

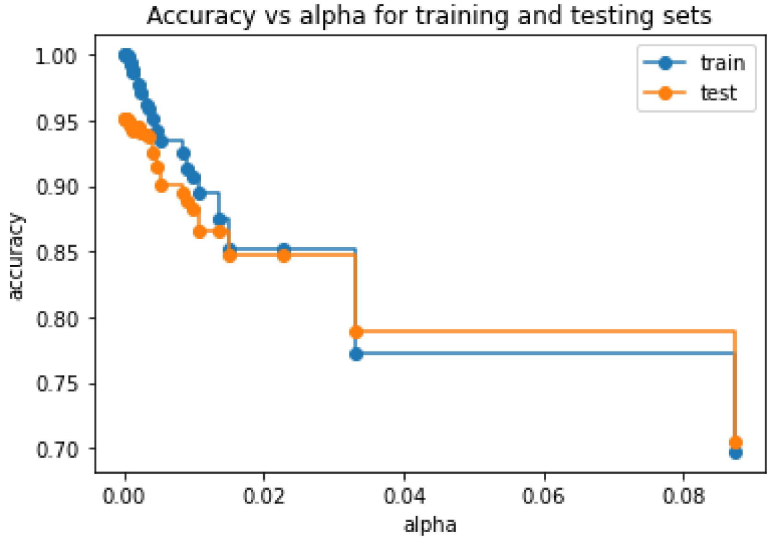
```
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0034212042639008935, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.004087383031272828, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.004637623574318561, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0051595204990417505, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.008421485662267494, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.008915931996522004, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.009910255327168393, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.010616975100101064, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.013679692722193944, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.014853478525823582, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.022731378437123598, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
```

```
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.033175884178617304, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.08738102722582908, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best')]
```

In [9]:

```
#EVALUATION OF ALPHA BASED ON TRAIN AND TEST ERRORS

#Evaluates prediction accuracy and plots it against ccp_alphas
train_scores = [dt.score(X_train, y_train) for dt in dts]
test_scores = [dt.score(X_test, y_test) for dt in dts]
fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



In [10]:

```
#FINAL MODEL BUILDING
dt=DecisionTreeClassifier(criterion='gini',
    min_samples_leaf=5 ,max_depth=10,
    ccp_alpha=0.0025)
dt.fit(X_train, y_train)
```

Out[10]:

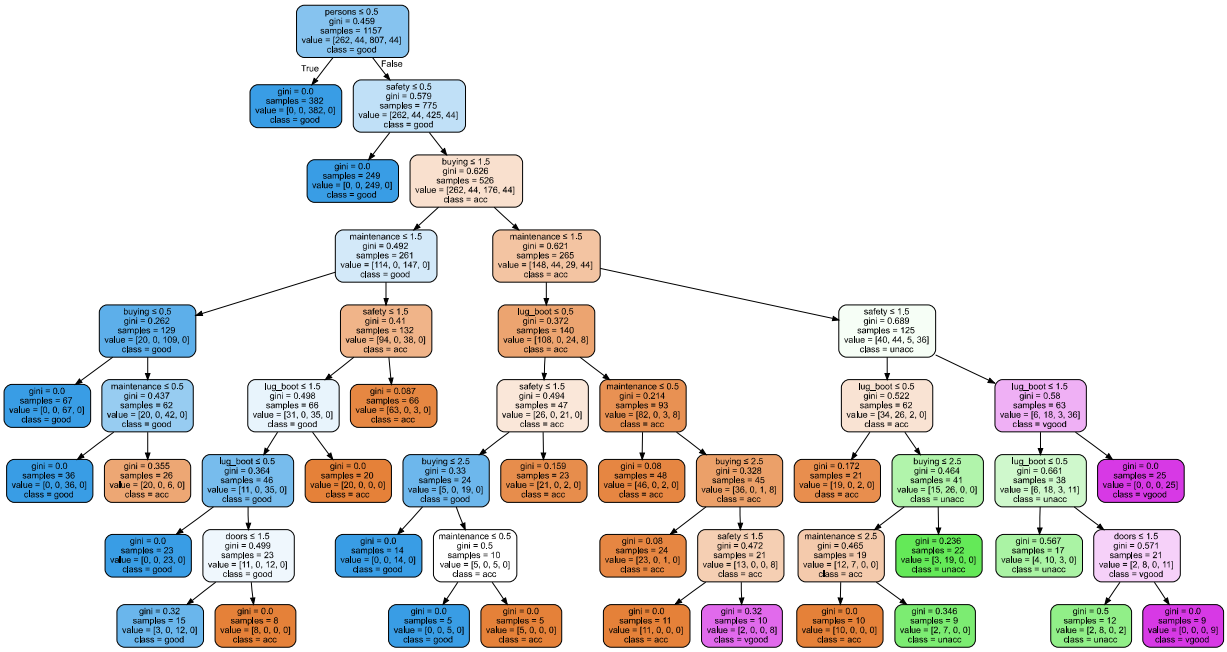
```
DecisionTreeClassifier(ccp_alpha=0.0025, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [11]:

```
#VISUALIZING THE TREE USING GRAPHVIZ
dot_data = tree.export_graphviz(dt, out_file=None,feature_names=X_train.columns,clas
```

```
graph = graphviz.Source(dot_data)
graph
#Scikit learn CART algorithm treats ordinal data also as integers and performs divis
```

Out[11]:



In [13]:

```
kfold = KFold(n_splits=5, random_state=10, shuffle=True)
X = X.to_numpy()
y = y.to_numpy()

no_classes = len(np.unique(y))

actual1 = np.empty([0], dtype=int)
predicted1 = np.empty([0], dtype=int)

for train_ndx, test_ndx in kfold.split(X,y):

    train_X, train_y, test_X, test_y = X[train_ndx], y[train_ndx], X[test_ndx], y[te

    actual1 = np.append(actual1, test_y)

    #dt.fit(train_X, train_y)
    predicted1 = np.append(predicted1, dt.predict(test_X))

#MODEL PREDICTION AND PERFORMANCE
y_pred=dt.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(actual1, predicted1))
cmtx = confusion_matrix(actual1, predicted1, labels=["acc","unacc", "good","vgood"])
cmtx
```

```
              precision    recall  f1-score   support

         acc       0.91      0.93      0.92       384
        good       0.72      1.00      0.84        69
       unacc       1.00      0.97      0.98      1210
       vgood       0.95      0.94      0.95        65

    accuracy                           0.96      1728
   macro avg       0.90      0.96      0.92      1728
weighted avg       0.97      0.96      0.96      1728
```

Out[13]:

```
array([[ 358,    4,   19,    3],
       [  34, 1172,    4,    0],
```

```
       [   0,    0,   69,    0],
       [   0,    0,    4,   61]], dtype=int64)
```

In [14]:

```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cmtx)
```

```
        FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
         0   1     2     4     3     4         5         6         7         8
acc     34  26  1310   358   384  1344  0.974702  0.965278  0.932292  0.034722
unacc    4  38   514  1172  1210   518  0.992278  0.975694  0.968595  0.024306
good    27   0  1632    69    69  1659  0.983725  0.984375  1.000000  0.015625
vgood    3   4  1660    61    65  1663  0.998196  0.995949  0.938462  0.004051
```

Bagging

In [15]:

```
#Bagging Classifier

from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier(50 , random_state= 10)

model = model.fit(X_train , y_train)

y_pred = model.predict(X_test)
```

In [16]:

```
kfold = KFold(n_splits=5, random_state=10, shuffle=True)
#X = X.to_numpy()
#y = y.to_numpy()

no_classes = len(np.unique(y))

actual2 = np.empty([0], dtype=int)
predicted2 = np.empty([0], dtype=int)

for train_ndx, test_ndx in kfold.split(X,y):

    train_X, train_y, test_X, test_y = X[train_ndx], y[train_ndx], X[test_ndx], y[te

    actual2 = np.append(actual2, test_y)
```

```
    #dt.fit(train_X, train_y)
    predicted2 = np.append(predicted2, dt.predict(test_X))



from sklearn.metrics import confusion_matrix , accuracy_score , classification_repor

cfm = confusion_matrix(actual2, predicted2, labels=["acc","unacc", "good","vgood"])
print(cfm)
print("Classification report :")

print(classification_report(actual2, predicted2, labels=["acc","unacc", "good","vgoo

acc = accuracy_score(actual2, predicted2)
print("Accuracy of the model:" , acc)
```

```
[[ 358    4   19    3]
 [  34 1172    4    0]
 [   0    0   69    0]
 [   0    0    4   61]]
Classification report :
              precision    recall  f1-score   support

         acc       0.91      0.93      0.92       384
       unacc       1.00      0.97      0.98      1210
        good       0.72      1.00      0.84        69
       vgood       0.95      0.94      0.95        65

    accuracy                           0.96      1728
   macro avg       0.90      0.96      0.92      1728
weighted avg       0.97      0.96      0.96      1728

Accuracy of the model: 0.9606481481481481
```

In [17]:

```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cfm)
```

```
        FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
         0   1     2     4     3     4         5         6         7         8
acc     34  26  1310   358   384  1344  0.974702  0.965278  0.932292  0.034722
unacc    4  38   514  1172  1210   518  0.992278  0.975694  0.968595  0.024306
good    27   0  1632    69    69  1659  0.983725  0.984375  1.000000  0.015625
vgood    3   4  1660    61    65  1663  0.998196  0.995949  0.938462  0.004051
```

Boosting

In [18]:
```
#boosting - Ada boost - generic boosting algo, less used for DT. can be used for oth

from sklearn.ensemble import AdaBoostClassifier

model_Ada = AdaBoostClassifier(base_estimator=DecisionTreeClassifier() ,n_estimators

model_Ada = model_Ada.fit(X_train , y_train)

y_pred = model_Ada.predict(X_test)
```

In [19]:
```
kfold = KFold(n_splits=5, random_state=10, shuffle=True)
#X = X.to_numpy()
#y = y.to_numpy()

no_classes = len(np.unique(y))

actual3 = np.empty([0], dtype=int)
predicted3 = np.empty([0], dtype=int)

for train_ndx, test_ndx in kfold.split(X,y):

    train_X, train_y, test_X, test_y = X[train_ndx], y[train_ndx], X[test_ndx], y[te

    actual3 = np.append(actual3, test_y)

    #dt.fit(train_X, train_y)
    predicted3 = np.append(predicted3, dt.predict(test_X))




from sklearn.metrics import confusion_matrix , accuracy_score , classification_repor

cfmt = confusion_matrix(actual3, predicted3, labels=["acc","unacc", "good","vgood"])
print(cfmt)
print("Classification report :")

print(classification_report(actual3, predicted3))

acc = accuracy_score(actual3, predicted3)
print("Accuracy of the model:" , acc)
```

```
[[ 358    4   19    3]
 [  34 1172    4    0]
 [   0    0   69    0]
 [   0    0    4   61]]
Classification report :
              precision    recall  f1-score   support

         acc       0.91      0.93      0.92       384
        good       0.72      1.00      0.84        69
       unacc       1.00      0.97      0.98      1210
       vgood       0.95      0.94      0.95        65

    accuracy                           0.96      1728
   macro avg       0.90      0.96      0.92      1728
weighted avg       0.97      0.96      0.96      1728

Accuracy of the model: 0.9606481481481481
```

In [20]:
```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cfmt)
```

```
        FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
         0   1     2     4     3     4         5         6         7         8
acc     34  26  1310   358   384  1344  0.974702  0.965278  0.932292  0.034722
unacc    4  38   514  1172  1210   518  0.992278  0.975694  0.968595  0.024306
good    27   0  1632    69    69  1659  0.983725  0.984375  1.000000  0.015625
vgood    3   4  1660    61    65  1663  0.998196  0.995949  0.938462  0.004051
```

Random forest

In [21]:
```
#RANDOM FOREST

dt=RandomForestClassifier(n_estimators=5, criterion='gini',
                          min_samples_leaf=5 ,max_depth=10,
                          ccp_alpha=0.0)
dt.fit(X_train, y_train)
#y_pred=dt.predict(X_test)
```

Out[21]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=5,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [22]:
```
kfold = KFold(n_splits=5, random_state=10, shuffle=True)



#y = y.to_numpy()

no_classes = len(np.unique(y))

actual4 = np.empty([0], dtype=int)
predicted4 = np.empty([0], dtype=int)

for train_ndx, test_ndx in kfold.split(X,y):
```

```
    train_X, train_y, test_X, test_y = X[train_ndx], y[train_ndx], X[test_ndx], y[te

    actual4 = np.append(actual4, test_y)

    #dt.fit(train_X, train_y)
    predicted4 = np.append(predicted4, dt.predict(test_X))




from sklearn.metrics import classification_report
cmtx_rn = confusion_matrix(actual4, predicted4, labels=["acc","unacc", "good","vgood
cmtx_rn
```

Out[22]:
```
array([[ 351,   24,    9,    0],
       [  20, 1190,    0,    0],
       [   8,    2,   58,    1],
       [  17,    0,   14,   34]], dtype=int64)
```

In [23]:
```
print(classification_report(y_test, y_pred, target_names=['acc','unacc','good','vgoo
```

```
              precision    recall  f1-score   support

         acc       0.89      0.89      0.89       122
       unacc       0.83      0.80      0.82        25
        good       0.98      0.98      0.98       403
       vgood       0.95      0.95      0.95        21

    accuracy                           0.95       571
   macro avg       0.91      0.90      0.91       571
weighted avg       0.95      0.95      0.95       571
```

In [24]:
```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cmtx_rn)
```

```
          FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
           0   1     2     4     3     4         5         6         7         8
acc       45  33  1299   351   384  1344  0.966518  0.954861  0.914062  0.045139
unacc     26  20   492  1190  1210   518  0.949807  0.973380  0.983471  0.026620
```

```
good    23   11  1636    58    69  1659  0.986136  0.980324  0.840580  0.019676
vgood    1   31  1662    34    65  1663  0.999399  0.981481  0.523077  0.018519
```

Comparision of various Decision tree model - Confusion Matrix using K-fold cross validation

In [25]:
```
print('CART')
ConfMat(cmtx)
```

```
CART
         FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
          0   1     2     4     3     4         5         6         7         8
acc      34  26  1310   358   384  1344  0.974702  0.965278  0.932292  0.034722
unacc     4  38   514  1172  1210   518  0.992278  0.975694  0.968595  0.024306
good     27   0  1632    69    69  1659  0.983725  0.984375  1.000000  0.015625
vgood     3   4  1660    61    65  1663  0.998196  0.995949  0.938462  0.004051
```

In [26]:
```
print('Bagging')
ConfMat(cfm)
```

```
Bagging
         FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
          0   1     2     4     3     4         5         6         7         8
acc      34  26  1310   358   384  1344  0.974702  0.965278  0.932292  0.034722
unacc     4  38   514  1172  1210   518  0.992278  0.975694  0.968595  0.024306
good     27   0  1632    69    69  1659  0.983725  0.984375  1.000000  0.015625
vgood     3   4  1660    61    65  1663  0.998196  0.995949  0.938462  0.004051
```

In [27]:
```
print('Boosting')
ConfMat(cfmt)
```

```
Boosting
         FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
          0   1     2     4     3     4         5         6         7         8
acc      34  26  1310   358   384  1344  0.974702  0.965278  0.932292  0.034722
unacc     4  38   514  1172  1210   518  0.992278  0.975694  0.968595  0.024306
good     27   0  1632    69    69  1659  0.983725  0.984375  1.000000  0.015625
vgood     3   4  1660    61    65  1663  0.998196  0.995949  0.938462  0.004051
```

In [28]:
```
print('RandomForest')
ConfMat(cmtx_rn)
```

```
RandomForest
         FP  FN    TN    TP     P     N       TNR       ACC       TPR       ERR
          0   1     2     4     3     4         5         6         7         8
acc      45  33  1299   351   384  1344  0.966518  0.954861  0.914062  0.045139
unacc    26  20   492  1190  1210   518  0.949807  0.973380  0.983471  0.026620
good     23  11  1636    58    69  1659  0.986136  0.980324  0.840580  0.019676
vgood     1  31  1662    34    65  1663  0.999399  0.981481  0.523077  0.018519
```

In [ ]:
```

```