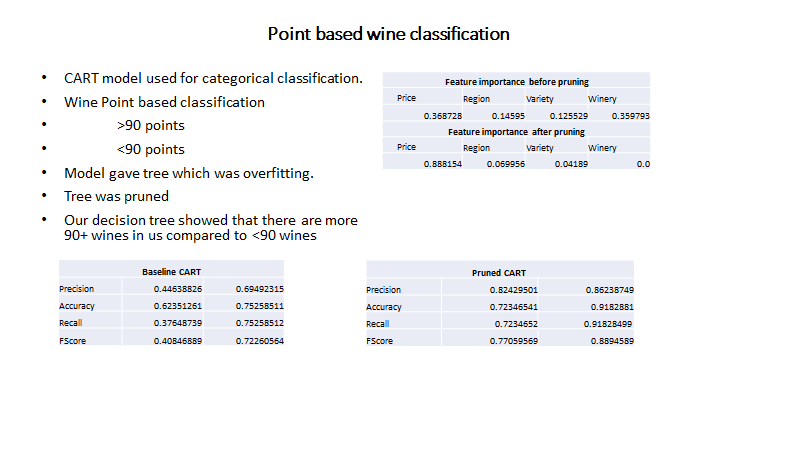
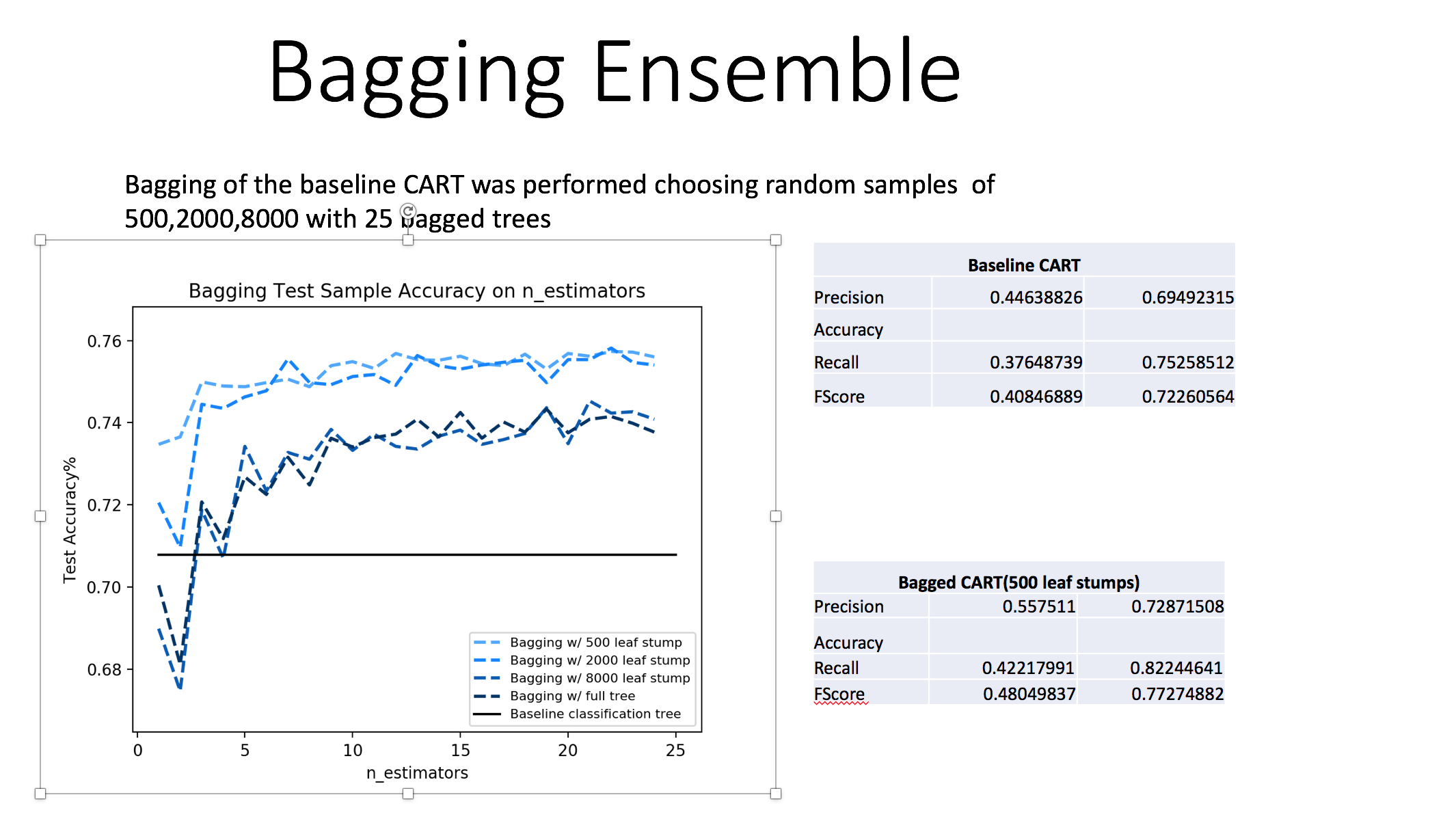
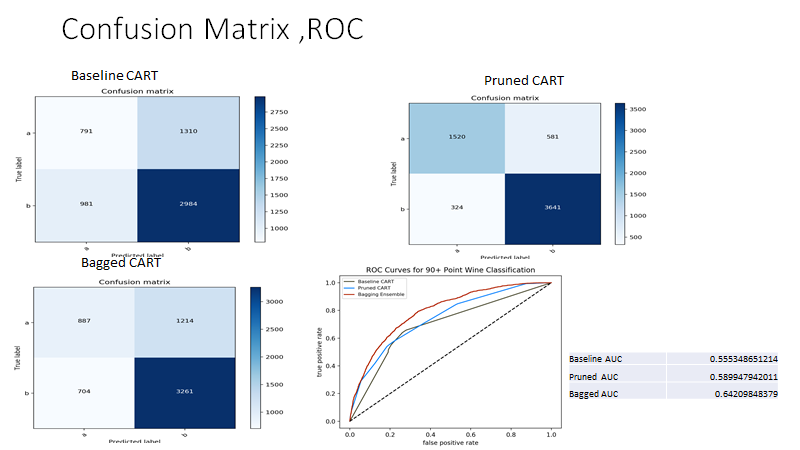
Rajeev George, Mariyam & Vasudevan, Chetan – Contribution





Confusion Matrix & ROC



Code:

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

import pandas as pd

from sklearn import metrics

from sklearn.externals.six import StringIO

from sklearn.metrics import accuracy\_score

import numpy as np

from sklearn.ensemble import AdaBoostClassifier

from matplotlib import pyplot as plt

from sklearn.tree import DecisionTreeClassifier, tree, export\_graphviz

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_curve, auc, confusion\_matrix

import pydotplus

import itertools

from sklearn.ensemble import BaggingClassifier

from sklearn.preprocessing import LabelEncoder, label\_binarize

# Read in & munge wine information dataset. US wines only

wine\_df=pd.read\_csv('/Users/bonythomas/new.csv')

wine\_df=wine\_df.loc[wine\_df.country=='US',['points','price','region\_1', 'variety2','winery2']]

wine\_df=wine\_df.dropna(axis=0,how='any')

print(wine\_df.columns)

# Map point values to categories

bin\_map={

100:'90+',

99:'90+',

98:'90+',

97:'90+',

96:'90+',

95:'90+',

94:'90+',

93:'90+',

92:'90+',

91:'90+',

90:'90+',

89:'<90',

88:'<90',

87:'<90',

86:'<90',

85:'<90',

84:'<90',

83:'<90',

82:'<90',

81:'<90',

80:'<90',

79:'<90',

78:'<90',

77:'<90',

76:'<90'}

wine\_df['point\_bins'] = wine\_df.points.map (bin\_map)

wine\_df.point\_bins.unique () # Ensure no records are un-binned

wine\_df = wine\_df.drop ('points', axis=1)

class\_names=['price','region', 'variety','winery']

obj\_df = wine\_df.select\_dtypes(include=['object']).copy()

wine\_df['point\_bins']=pd.factorize(obj\_df['point\_bins'])[0]

NintyAbove=wine\_df.loc[wine\_df.point\_bins==0]

NintyBelow=wine\_df.loc[wine\_df.point\_bins==1]

print("NintyAbove",NintyAbove.shape)

print("NintyBelow",NintyBelow.shape)

# Split into 50/50 train/test datasets

wine\_train, wine\_test = train\_test\_split (wine\_df, test\_size=0.15)

print("Wine Training data shape:",wine\_train.shape)

print("Wine Testing data shape:",wine\_test.shape)

# Prepare train data for classification tree

regn\_lab = LabelEncoder ().fit (np.unique (wine\_df.region\_1.values))

var\_lab = LabelEncoder ().fit (np.unique (wine\_df.variety2.values))

wnry\_lab = LabelEncoder ().fit (np.unique (wine\_df.winery2.values))

wine\_train['regn\_enc'] = regn\_lab.transform (wine\_train.region\_1)

wine\_train['var\_enc'] = var\_lab.transform (wine\_train.variety2)

wine\_train['wnry\_enc'] = wnry\_lab.transform (wine\_train.winery2)

wine\_train = wine\_train.drop (['region\_1', 'variety2', 'winery2'], axis=1)

target=['points']

# Prepare test data for classification tree

wine\_test['regn\_enc'] = regn\_lab.transform (wine\_test.region\_1)

wine\_test['var\_enc'] = var\_lab.transform (wine\_test.variety2)

wine\_test['wnry\_enc'] = wnry\_lab.transform (wine\_test.winery2)

wine\_test = wine\_test.drop (['region\_1', 'variety2', 'winery2'], axis=1)

# Train classification tree

x = wine\_train.loc[:, ['price', 'regn\_enc', 'var\_enc', 'wnry\_enc']]

y = wine\_train['point\_bins']

x\_test = wine\_test.loc[:, ['price', 'regn\_enc', 'var\_enc', 'wnry\_enc']]

y\_test = wine\_test['point\_bins']

min\_samp\_split = 2

clf = DecisionTreeClassifier (min\_samples\_split=min\_samp\_split, max\_features=None)

clf1 = clf.fit (x, y)

# BASELINE - all features, no tree termination criteria

train\_error = y == clf.predict (x)

test\_error1 = wine\_test['point\_bins'] == clf.predict (wine\_test.loc[:, ['price', \

'regn\_enc', 'var\_enc', 'wnry\_enc']])

print ('@@@@@@@@@@@@@@@@@@@@@@@@@')

print ('CART w/ #leaf nodes = ', clf.tree\_.node\_count)

print (' ', clf.n\_features\_, ' features out of: 4 features')

print (' training accuracy: ', '{:.1%}'.format (sum (train\_error) / len (

train\_error)))

print (' test accuracy: ', '{:.1%}'.format (sum (test\_error1) / len (test\_error1)))

# Report feature importance

print ('@@@@@@@@@@@@@@@@@@@@@@@@@')

print ('Feature importance for max\_leaves model')

print (pd.DataFrame ([clf.feature\_importances\_], columns=x.columns.values))

#We will rebuild a new tree by using above data and see how it works by tweeking the parameteres

x = wine\_train.loc[:, ['price', 'regn\_enc', 'var\_enc', 'wnry\_enc']]

y = wine\_train['point\_bins']

dtree = tree.DecisionTreeClassifier(criterion = "gini", splitter = 'random', max\_leaf\_nodes = 20, min\_samples\_leaf = 5, max\_depth= 5)

dtree.fit(x, y)

prunetrain\_error = y == dtree.predict (x)

dot\_data = StringIO()

export\_graphviz(dtree, out\_file=dot\_data,feature\_names=class\_names,

filled=True, rounded=True,

special\_characters=True)

export\_graphviz(dtree, out\_file='wineDT1Pruned.dot')

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png("WineDTPruned.png")

print("Feature importance after pruning:")

print (pd.DataFrame ([dtree.feature\_importances\_], columns=x.columns.values))

Prunetest\_error1 = wine\_test['point\_bins'] == dtree.predict(wine\_test.loc[:, ['price','regn\_enc', 'var\_enc', 'wnry\_enc']])

print ('@@@@@@@@@@@@@@@@@@@@@@@@@')

print ('CART after pruning w/ #leaf nodes = ', dtree.tree\_.node\_count)

from sklearn.metrics import precision\_recall\_fscore\_support as score

predicted = test\_error1

y\_test = y\_test

precision, recall, fscore, support = score(y\_test, predicted)

print('precision before pruning: {}'.format(precision))

print('recall before pruning: {}'.format(recall))

print('fscore before pruning: {}'.format(fscore))

print('support before pruning: {}'.format(support))

from sklearn.metrics import precision\_recall\_fscore\_support as score

precision1, recall1, fscore1, support1 = score(y\_test, Prunetest\_error1)

print('precision after pruning: {}'.format(precision1))

print('recall after pruning: {}'.format(recall1))

print('fscore after pruning: {}'.format(fscore1))

print('support after pruning: {}'.format(support1))

# Control the number of n\_estimators in ensemble functions

max\_n\_ests=25

# Create dataframe to record results of ensembles.

results=pd.DataFrame([],columns=list(['type','n\_leaf','n\_est', \

'train\_acc','test\_acc']))

# Train bagging ensemble on iterations of n\_estimators=i

# and iterations of stump max\_leaf\_nodes=j

for j in [20,29,39]:

clf\_stump=DecisionTreeClassifier(criterion = "gini", splitter = 'random', max\_leaf\_nodes = j, min\_samples\_leaf = 5, max\_depth= 5)

for i in np.arange(1,max\_n\_ests):

baglfy=BaggingClassifier(base\_estimator=clf\_stump,n\_estimators=i,

max\_samples=1.0)

baglfy=baglfy.fit(x,y)

bag\_tr\_err=y==baglfy.predict(x)

bag\_tst\_err=wine\_test['point\_bins']==baglfy.predict( \

wine\_test.loc[:,['price','regn\_enc','var\_enc','wnry\_enc']])

run\_rslt=pd.DataFrame([['bag',j,i,sum(bag\_tr\_err)/len(bag\_tr\_err),

sum(bag\_tst\_err)/len(bag\_tst\_err)]],

columns=list(['type','n\_leaf','n\_est','train\_acc','test\_acc']))

results=results.append(run\_rslt)

# Train bagging ensemble on iterations of n\_estimators=i

# and iterations of stump max\_leaf\_nodes=j

# Train boosting ensemble on iterations of n\_estimators=i

# and iterations of stump max\_leaf\_nodes=j

for j in [20,29,39]:

clf\_stump=DecisionTreeClassifier(criterion = "gini", splitter = 'random', max\_leaf\_nodes = j, min\_samples\_leaf = 5, max\_depth= 5)

for i in np.arange(1,max\_n\_ests):

print(i)

bstlfy=AdaBoostClassifier(base\_estimator=clf\_stump,n\_estimators=i)

bstlfy=bstlfy.fit(x,y)

bst\_tr\_err=y==bstlfy.predict(x)

bst\_tst\_err=wine\_test['point\_bins']==bstlfy.predict( \

wine\_test.loc[:,['price','regn\_enc','var\_enc','wnry\_enc']])

run\_rslt=pd.DataFrame([['bst',j,i,sum(bst\_tr\_err)/len(bst\_tr\_err),

sum(bst\_tst\_err)/len(bst\_tst\_err)]],

columns=list(['type','n\_leaf','n\_est','train\_acc','test\_acc']))

results=results.append(run\_rslt)

# ROC curve for baseline classification tree

clf\_probs=clf1.predict\_proba(wine\_test.loc[:,['price','regn\_enc','var\_enc', \

'wnry\_enc']])

fpr1,tpr1,thr1=roc\_curve(np.where(wine\_test['point\_bins']==0,1.,0.), \

clf\_probs[:,0])

clf\_probs=dtree.predict\_proba(wine\_test.loc[:,['price','regn\_enc','var\_enc', \

'wnry\_enc']])

fpr2,tpr2,thr2=roc\_curve(np.where(wine\_test['point\_bins']==0,1.,0.), \

clf\_probs[:,0])

# ROC curve for bagging ensemble using full classification trees

bag\_probs=baglfy.predict\_proba(wine\_test.loc[:,['price','regn\_enc', \

'var\_enc','wnry\_enc']])

fpr3,tpr3,thr3=roc\_curve(np.where(wine\_test['point\_bins']==0,1.,0.), \

bag\_probs[:,0])

# ROC curve for boosting ensemble using full classification trees

boost\_probs=bstlfy.predict\_proba(wine\_test.loc[:,['price','regn\_enc', \

'var\_enc','wnry\_enc']])

fpr4,tpr4,thr4=roc\_curve(np.where(wine\_test['point\_bins']==0,1.,0.), \

boost\_probs[:,0])

# Plot ROC Curves

plt.plot(fpr1,tpr1,color='#4d4d33',label='Baseline CART')

plt.plot(fpr2,tpr2,color='#0080ff',label='Pruned CART')

plt.plot(fpr3,tpr3,color='#b32400',label='Bagging Ensemble')

plt.plot(fpr4,tpr4,color='#661400',label='Boosting Ensemble')

plt.plot([0.,1.],[0.,1.],color='k',linestyle='--')

plt.title('ROC Curves for 90+ Point Wine Classification')

plt.xlabel('false positive rate')

plt.ylabel('true positive rate')

plt.legend(fontsize=8)

plt.show()

#Baseline Classification Pruned Tree Perfromance

from sklearn.metrics import precision\_recall\_fscore\_support as score

precision, recall, fscore, support = score(wine\_test['point\_bins'], Prunetest\_error1)

print('baseline precision: {}'.format(precision))

print('baseline recall: {}'.format(recall))

print('baseline fscore: {}'.format(fscore))

print('baseline support: {}'.format(support))

#Baseline Classification Pruned Tree Perfromance after bagging

precision2, recall2, fscore2, support2 = score(wine\_test['point\_bins'], bst\_tst\_err)

print('precision after boosting: {}'.format(precision2))

print('recall after boosting: {}'.format(recall2))

print('fscore after boosting: {}'.format(fscore2))

print('support after boosting: {}'.format(support2))

precision1, recall1, fscore1, support1 = score(wine\_test['point\_bins'], bag\_tst\_err)

print('precision after bagging: {}'.format(precision1))

print('recall after bagging: {}'.format(recall1))

print('fscore after bagging: {}'.format(fscore1))

print('support after bagging: {}'.format(support1))

cm = confusion\_matrix(y\_test.values, test\_error1)

cm1 = confusion\_matrix(y\_test.values, Prunetest\_error1)

cm2 = confusion\_matrix(y\_test.values, bag\_tst\_err)

cm3 = confusion\_matrix(y\_test.values, bst\_tst\_err)

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.figure()

plot\_confusion\_matrix(cm, ['a', 'b'])

plt.figure()

plot\_confusion\_matrix(cm1, ['a', 'b'])

plt.figure()

plot\_confusion\_matrix(cm2, ['a', 'b'])

plt.figure()

plot\_confusion\_matrix(cm3, ['a', 'b'])

plt.show()

test\_labels=['90+ Wine','<90 Wine']

# Calculate Accuracy Rate by using accuracy\_score()

print("The accuracy score of baseline CART is :",accuracy\_score(wine\_test['point\_bins'], test\_error1))

print("Accuracy of pruned CART:",accuracy\_score(wine\_test['point\_bins'], Prunetest\_error1))

print("Accuracy of bagged CART:",accuracy\_score(wine\_test['point\_bins'], bag\_tst\_err))

print("Accuracy of boosted CART:",accuracy\_score(wine\_test['point\_bins'], bst\_tst\_err))

Results:

@@@@@@@@@@@@@@@@@@@@@@@@@

CART w/ #leaf nodes = 18949

4 features out of: 4 features

training accuracy: 94.0%

test accuracy: 70.2%

@@@@@@@@@@@@@@@@@@@@@@@@@

Feature importance for max\_leaves model

price regn\_enc var\_enc wnry\_enc

0 0.368728 0.14595 0.125529 0.359793

@@@@@@@@@@@@@@@@@@@@@@@@@

Feature importance after pruning:

price regn\_enc var\_enc wnry\_enc

0 0.876723 0.123277 0.0 0.0

@@@@@@@@@@@@@@@@@@@@@@@@@

CART after pruning w/ #leaf nodes = 39

precision before pruning: [ 0.41218837 0.70053978]

recall before pruning: [ 0.36831683 0.73776569]

fscore before pruning: [ 0.38901961 0.718671 ]

support before pruning: [2020 4046]

@@@@@@@@@@@@@@@@@@@@@@@@@

precision after pruning: [ 0.70664207 0.80382883]

recall after pruning: [ 0.56881188 0.88210578]

fscore after pruning: [ 0.63027976 0.84115013]

support after pruning: [2020 4046]

@@@@@@@@@@@@@@@@@@@@@@@@@

baseline precision: [ 0.70664207 0.80382883]

baseline recall: [ 0.56881188 0.88210578]

baseline fscore: [ 0.63027976 0.84115013]

baseline support: [2020 4046]

@@@@@@@@@@@@@@@@@@@@@@@@@

precision after boosting: [ 0.57261676 0.75016656]

recall after boosting: [ 0.44306931 0.83489867]

fscore after boosting: [ 0.49958136 0.79026787]

support after boosting: [2020 4046]

@@@@@@@@@@@@@@@@@@@@@@@@@

precision after bagging: [ 0.73489519 0.81368137]

recall after bagging: [ 0.59009901 0.89372219]

fscore after bagging: [ 0.65458539 0.85182568]

support after bagging: [2020 4046]

@@@@@@@@@@@@@@@@@@@@@@@@@

Confusion matrix Baseline

[[ 744 1276]

[1061 2985]]

@@@@@@@@@@@@@@@@@@@@@@@@@

Confusion matrix Pruned

[[1149 871]

[ 477 3569]]

@@@@@@@@@@@@@@@@@@@@@@@@@

Confusion matrix Bagged

[[1192 828]

[ 430 3616]]

@@@@@@@@@@@@@@@@@@@@@@@@@

Confusion matrix Boosted

[[ 895 1125]

[ 668 3378]]

@@@@@@@@@@@@@@@@@@@@@@@@@

The accuracy score of baseline CART is : 0.614737883284

Accuracy of pruned CART: 0.777777777778

Accuracy of bagged CART: 0.79261457303

Accuracy of boosted CART: 0.70441806792