**IN498 Unit 4 Laurence Burden**

**Summarize what the area under the receiver operating characteristic curve is. Include when and why it is used.**

ROC is an analysis for evaluating the accuracy of a statistical model. Specifically, the model must output a binary classification. This means it is good at measuring analysis of information such as disease diagnosis. The area under the curve acts as an overall summary of diagnostic accuracy. An AUC of 0.5 is equal to random chance and 1.0 represents perfect accuracy.

**What is the mean absolute percentage error result?**

The MAPE result is -infinity.

**Summarize what the mean absolute percentage error is and discuss a weakness of it.**

The MAPE uses each error percentage from all of the observations and predictions to find the mean of the overall errors. This method struggles to provide meaningful information if there are zeros in the error list. This causes a divide by zero error for the overall mean.

**Python Code Output**

Decision Tree Accuracy with Training Data: (0.8942851570118978,)

Decision Tree Accuracy with Test Data: (0.8689547581903276,)

Random Forest Prediction: (array([0, 0, 0, ..., 0, 0, 1]),)

Random Forest Accuracy: (0.8689547581903276,)

Random Forest Prediction of Keeping 1 User 30 Days with 1 Install: (array([0]),)

Random Forest Prediction of Keeping 1 User 30 Days with 2 Install: (array([1]),)

Random Forest Prediction of Keeping 1 User 30 Days with 4 Install: (array([1]),)

Absolute Errors for Random Forest Model: (5022 0

4926 0

1959 0

894 1

113 0

..

553 0

818 0

2957 0

3576 1

3726 1

Name: Install\_30, Length: 1282, dtype: int32,)

Mean Absolute Error: (0.13,)

Random Forest Probabilities X\_test(array([3.04589002e-04, 3.04589002e-04, 3.04589002e-04, ...,

3.04589002e-04, 3.23034652e-01, 5.49965852e-01]),)

ROC AUC for Random Forest: (0.9211627906976745,)

Random Forest MAPE(5022 NaN

4926 NaN

1959 NaN

894 inf

113 NaN

...

553 NaN

818 NaN

2957 NaN

3576 100.0

3726 inf

Name: Install\_30, Length: 1282, dtype: float64,)

Random Forest Accuracy Percentage Using Error Percentage: (-inf,)

**Python Code**

import pandas as pd  
import numpy as np  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import roc\_auc\_score  
from sklearn import metrics  
import sys  
  
# Ignoring warnings  
if not sys.warnoptions:  
 import warnings  
 warnings.simplefilter("ignore")  
  
  
# Set the writeFunction boolean  
# True = write to concole  
# False = write to file for assignment turnin  
PRINT = False  
  
# Open a file handle for assignment results.  
if PRINT == False:  
 f = open("IN498\_Unit4\_Burden.txt", "a")  
###############################################  
#  
# PURPOSE: Write to console or file based on  
# the writeFunction variable  
# True = write to console  
# False = write to file for assignment turnin  
#  
# INPUT: Message to write (message1)  
# Optional: message2  
#  
# OUTPUT: None  
#  
###############################################  
def writeFunction(message1, \*message2):  
  
 # Print to console  
 if PRINT:  
 print(message1)  
 print(message2)  
 print()  
 # Print to file for assignment  
 else:  
 f.write(str(message1))  
 f.write(str(message2))  
 f.write("\n\n")  
  
  
# Widen the column display  
pd.set\_option('max\_colwidth', 500)  
  
# Read data into a DataFrame using these columns  
# "Date","Package\_Name","Country","Store\_Listing\_Visitors",  
# "Installers","Visitor-to-Installer\_conversion\_rate",  
# "Installers\_retained\_for\_1\_day","Installer-to-1\_day\_retention\_rate",  
# "Installers\_retained\_for\_7\_days","Installer-to-7\_days\_retention\_rate",  
# "Installers\_retained\_for\_15\_days","Installer-to-15\_days\_retention\_rate",  
# "Installers\_retained\_for\_30\_days","Installer-to-30\_days\_retention\_rate"  
col\_names = ["Date", "Package\_Name", "Country", "Store\_Listing\_Visitors",  
 "Installers", "Visitor-to-Installer\_conversion\_rate", "Installers\_retained\_for\_1\_day",  
 "Installer-to-1\_day\_retention\_rate", "Installers\_retained\_for\_7\_days",  
 "Installer-to-7\_days\_retention\_rate", "Installers\_retained\_for\_15\_days",  
 "Installer-to-15\_days\_retention\_rate", "Installers\_retained\_for\_30\_days",  
 "Installer-to-30\_days\_retention\_rate"]  
data = pd.read\_csv('final\_retentions\_parsed.csv', names=col\_names)  
  
############ FIX MISSING DATA #######################  
# Replace NaN with 0 for Installers\_retained\_for\_30\_days  
data[np.isnan(data.Installers\_retained\_for\_30\_days)] = 0  
  
#################################### INSTALL\_30 ##########################################  
# Add a new column for installers retained for 30 days  
# If greater than 0, put 1, if 0, put 0  
data['Install\_30'] = np.where(data['Installers\_retained\_for\_30\_days'] > 0, 1, 0)  
  
# Create X using Installers and y using Install\_30 columns  
X = pd.DataFrame({'Installers': data['Installers']})  
y = pd.DataFrame({'Install\_30': data['Install\_30']})  
  
# print('data shape: ', data.shape)  
# print('shape: ', data['Installers'].shape)  
# print('X Shape: ', X.shape)  
# print('y Shape: ', y.shape)  
  
# Train/test split 80% train, 20% test  
# Save into these variables: X\_train, X\_test, y\_train, y\_test  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2)  
  
########### DECISION TREE #######################  
# Build a decision tree model  
dt = DecisionTreeClassifier()  
  
# Fit the tree using X\_train and y\_train  
dt.fit(X\_train, y\_train)  
  
# Print the accuracy of the tree using X\_train and y\_train  
writeFunction('Decision Tree Accuracy with Training Data: ', dt.score(X\_train, y\_train))  
  
writeFunction('Decision Tree Accuracy with Test Data: ', dt.score(X\_test, y\_test))  
  
########### RANDOM FOREST #######################  
#Build a random forest tree model  
# n\_estimators = 1000  
rf = RandomForestClassifier(n\_estimators=1000)  
  
#Fit the random tree model with X\_train and y\_train  
rf.fit(X\_train, y\_train)  
  
#Predict using X\_test and the random forest model and store in y\_pred  
y\_pred = rf.predict(X\_test)  
  
#Print the random forest results using y\_pred  
writeFunction('Random Forest Prediction: ', y\_pred)  
  
#Print the accuracy for the random forest model using y\_test and y\_pred  
writeFunction('Random Forest Accuracy: ', metrics.accuracy\_score(y\_test, y\_pred))  
  
#Predict retention over 30 days for number of installs  
# Use 1,2,4  
writeFunction('Random Forest Prediction of Keeping 1 User 30 Days with 1 Install: ',  
 rf.predict([[1]]))  
writeFunction('Random Forest Prediction of Keeping 1 User 30 Days with 2 Install: ',  
 rf.predict([[2]]))  
writeFunction('Random Forest Prediction of Keeping 1 User 30 Days with 4 Install: ',  
 rf.predict([[4]]))  
  
# Get the absolute errors for the random forest model  
# Use y\_pred and y\_test  
errors = abs(y\_pred - y\_test['Install\_30'])  
  
# Print the absolute errors for the random forest model  
# Use y\_pred and y\_test  
writeFunction('Absolute Errors for Random Forest Model', errors)  
  
# Print out the mean absolute error  
writeFunction('Mean Absolute Error', round(np.mean(errors), 2))  
  
# Get the predictions using X\_test and save to rf\_probs  
rf\_probs = rf.predict\_proba(X\_test)[:, 1]  
  
# Print the predictions for Random Forest using rf\_probs  
writeFunction('Random Forest Probabilities X\_test', rf\_probs)  
  
# Compute Area Under the Receiver Operating Characteristic Curve  
# Get the ROC AUC score for the random forest model for y\_test and rf\_probs  
roc\_value = roc\_auc\_score(y\_test, rf\_probs)  
  
# Print the ROC AUC for the random forest model  
writeFunction('ROC AUC for Random Forest', roc\_value)  
  
# Get the mean absolute percentage error (MAPE) using y\_test  
mape = 100 \* (errors / y\_test['Install\_30'])  
  
# Print the mean absolute percentage error (MAPE) using y\_test  
writeFunction('Random Forest MAPE', mape)  
  
# Get the accuracy for the random forest model using mape  
accuracy = 100 - np.mean(mape)  
  
# Print the accuracy for the random forest model  
writeFunction('Random Forest Accuracy Percentage Using Error Percentage', accuracy)  
  
# Close the file handle  
if PRINT == False:  
 f.close()