STAT303-3 Assignment 3 Complete

May 2, 2022

Instructions:

a. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.

b. Do not write your name on the assignment. (1 point)

c. Export your Jupyter notebook as a PDF file. If you get an error, make sure you have downloaded the MikTex software (for windows) or MacTex (for mac). Note that after installing MikTex/MacTex, you will need to check for updates, install the updates if needed, and re-start your system. Submit the PDF file. (1 point)

d. Please include each question (or question number) followed by code and your answer (if applicable). Write your code in the 'Code' cells and your answer in the 'Markdown' cells of the Jupyter notebook. Ensure that the solution is well-organized, clear, and concise (3 points)

1. It's easy enough to identify different sections of the homework assignment (e.g., if there are different sections of an assignment, they're clearly distinguishable by section headers or the like)

2. It's clear which code/markdown blocks correspond to which questions.
3. There aren't excessively long outputs of extraneous information (e.g., no printouts of entire data frames without good reason)

This assignment is **due at 11:59pm on Wednesday, May 11th**. Good luck!

Submissions will be graded with a maximum of 40 points – 35 points for code & answers, 5 points for anonymity and proper formatting. However, your final grade in the assignment will be scaled to be out of 100 points. For example, if you scored 30/40 in the assigment, your score will be scaled to 75/100

Q1a

Develop a bagged decision tree model on the data *house_feature_train.csv* to predict *house_price*. Use all the predictors except *house_id*. Follow the steps below:

(i) Make a plot of out-of-bag RMSE (Root mean squared error) vs number of trees. *(3 points for code, 1 point for answer)*

(ii) Based on the plot in (i), develop a bagged tree model with an appropriate number of trees. *(2 points for code)*

(iii) Use the model developed in (ii) to predict house prices in *house_feature_test.csv*. Report the RMSE (should be less than 350). *(1 point for code, 1 point for answer)*

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.metrics import␣
 ↪mean_squared_error,r2_score,roc_curve,auc,precision_recall_curve
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, ParameterGrid
from sklearn.ensemble import␣
 ↪BaggingRegressor,BaggingClassifier,RandomForestRegressor,RandomForestClassifier
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
import itertools as it
import time as time
```

[2]:
```
trainf = pd.read_csv('house_feature_train.csv')
trainp = pd.read_csv('house_price_train.csv')
testf = pd.read_csv('house_feature_test.csv')
testp = pd.read_csv('house_price_test.csv')
train = pd.merge(trainf,trainp)
test = pd.merge(testf,testp)
train.head()
```

[2]:

	house_id	house_age	distance_MRT	number_convenience_stores	latitude	\
0	210	5.2	390.5684	5	24.97937	
1	190	35.3	616.5735	8	24.97945	
2	328	15.9	1497.7130	3	24.97003	
3	5	7.1	2175.0300	3	24.96305	
4	412	8.1	104.8101	5	24.96674	

	longitude	house_price
0	121.54245	2724.84
1	121.53642	1789.29
2	121.51696	556.96
3	121.51254	1030.41
4	121.54067	2756.25

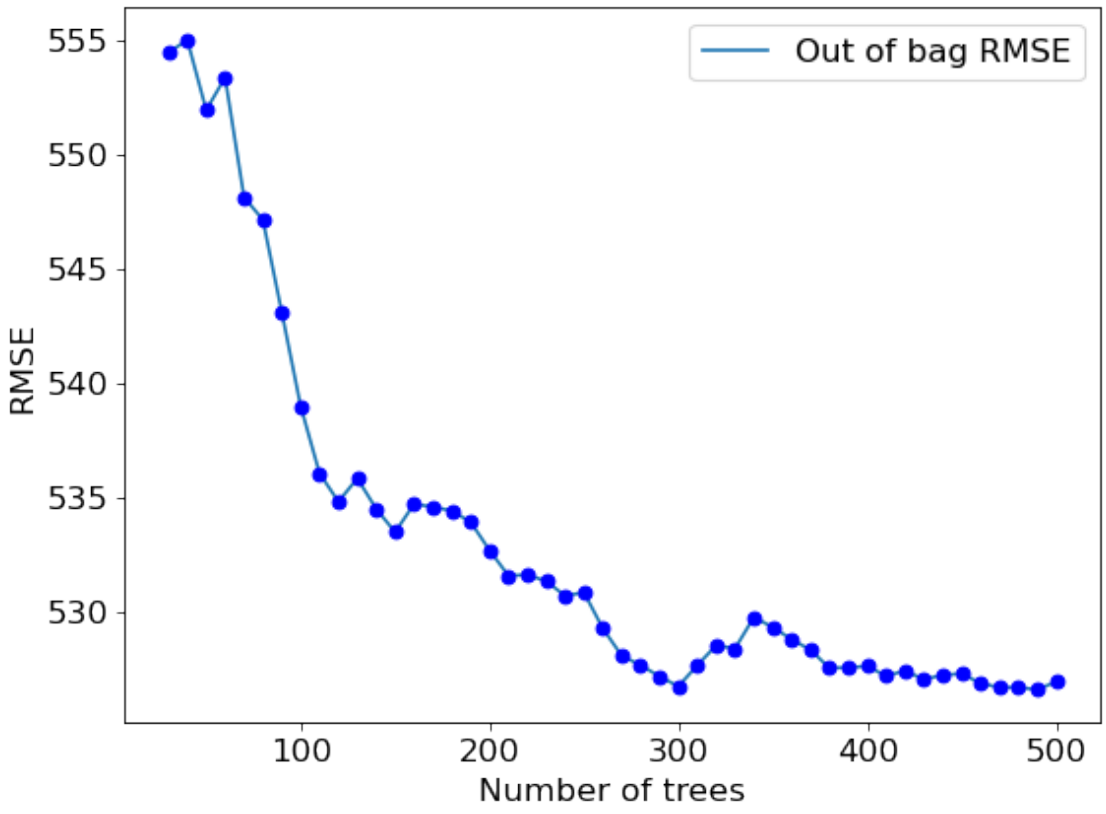
[3]:
```
X =␣
 ↪train[['house_age','distance_MRT','number_convenience_stores','latitude','longitude']]
Xtest =␣
 ↪test[['house_age','distance_MRT','number_convenience_stores','latitude','longitude']]
y = train['house_price']
ytest = (test['house_price'])
```

[4]:
```
#Finding OOB model RMSE vs number of trees
start_time = time.time()
oob_rmse={};
for i in np.linspace(30,500,48,dtype=int):
    model = BaggingRegressor(base_estimator=DecisionTreeRegressor(),␣
 ↪n_estimators=i, random_state=1,
                         n_jobs=-1,oob_score=True).fit(X, y)
    oob_rmse[i]=np.sqrt(mean_squared_error(model.oob_prediction_,y))
end_time = time.time()-start_time
print(end_time/60, " minutes")
```

```
0.13434866666793824  minutes
```

[5]:
```
#Visualizing out-of-bag RMSE
plt.rcParams.update({'font.size': 15})
plt.figure(figsize=(8, 6), dpi=80)
plt.plot(oob_rmse.keys(),oob_rmse.values(),label = 'Out of bag RMSE')
plt.plot(oob_rmse.keys(),oob_rmse.values(),'o',color = 'blue')
plt.xlabel('Number of trees')
plt.ylabel('RMSE')
plt.legend()
```

[5]:
```
<matplotlib.legend.Legend at 0x7fb2e8045130>
```



Based on the above plot, a model with 300 trees seems to suffice for minimizing the OOB RMSE.

[6]:
```
#Bagging with 300 trees
model = BaggingRegressor(base_estimator=DecisionTreeRegressor(),␣
 ↪n_estimators=300, random_state=1,
                         oob_score=False,n_jobs=-1).fit(X, y)
#RMSE on test data
pred = model.predict(Xtest)
print("RMSE = ",np.sqrt(mean_squared_error(test.house_price, pred)))
```

```
RMSE =  341.0988435793935
```

Q1b

Develop a random forest on the data *house_feature_train.csv* to predict *house_price*. Use all the predictors except *house_id*. For tuning the model, compute the out-of-bag score (OOB R-squared) for the following set of parameter values:

(i) Number of trees = [75,100,125]

(ii) Number of predictors considered at each split: [1,2,3,4,5],

(iii) Minimum number of observations required in a non-terminal node to split it: [2,3,4,5,6],

(iv) Minimum number of observations required in a leaf: [1,2,3]

Use the parameter values corresponding to the model with the highest OOB score to develop the model. Use the developed model to predict house prices in *house_feature_test.csv*. Report the RMSE (should be less than the RMSE in *Q1a*).

(7 points for code, 1 point for answer)

[7]:
```
start_time = time.time()

n_samples = train.shape[0]
n_features = train.shape[1]

params = {'n_estimators': [75,100,125],
          'max_features': [1,2,3,4,5],
          'min_samples_split':[2,3,4,5,6],
          'min_samples_leaf':[1,2,3]}

param_list = list(it.product(*(params[Name] for Name in list(params.keys()))))

oob_score = [0]*len(param_list)
i=0
for pr in param_list:
    model =␣
 ↪RandomForestRegressor(random_state=1,oob_score=True,verbose=False,n_estimators␣
 ↪= pr[0],
                                    max_features=pr[1],
                                ␣
 ↪min_samples_split=pr[2],min_samples_leaf=pr[3],
                                    n_jobs=-1).fit(X,y)
    oob_score[i] = model.oob_score_
    i=i+1

end_time = time.time()
print("time taken = ", (end_time-start_time)/60, " minutes")
print("max score = ", np.max(oob_score))
print("best params= ", param_list[np.argmax(oob_score)])
```

```
time taken =  0.3605940024058024  minutes
max score =  0.7660491688534226
best params=  (100, 3, 5, 2)
```

[8]:
```
#Random forest with 100 trees
model = RandomForestRegressor(random_state=1,n_estimators=100, max_features=3,
                              min_samples_split=5,min_samples_leaf=2,n_jobs=-1).
 ↪fit(X, y)
#RMSE on test data
```

```
pred = model.predict(Xtest)
print("RMSE = ", np.sqrt(mean_squared_error(test.house_price, pred)))
```

```
RMSE =  326.42169817343915
```

Q2

Refer to *Q2a of assignment 2*. Make an attempt to tune a random forest model (instead of a single decision tree model as in *Q2a of assignment 2*) that has both **precision and recall higher than 40%** on *train.csv*, *test1.csv* and *test2.csv*. Attempt to tune the number of trees (`n_estimators`), and the number of predictors to consider at each split (`max_features`) to achieve the objective, but do not tune any parameters that prune trees (i.e., do not tune `max_depth`, `max_leaf_nodes`, `min_samples_split`, and `min_samples_leaf`, etc.).

Tune the parameters as follows. Compute the out-of-bag (OOB) precision for different parameter values of `n_estimators`, `max_features` and/or any other parameters that do not prune the tree, and choose the parameter values corresponing to the maximum OOB precision. Show that you fail to obtain a model that meets the objective. You can show that by making a precision-recall plot for the optimized model, where both precision and recall are never simultaneously greater than 40%.

Purpose of this exercise: This exercise shows that pruning a single decision tree model may sometimes (though not usually) lead to a higher reduction in prediction variance (or a more accurate model), as compared to a random forest with unpruned trees. Thus,

(i) A random forest model can sometimes be further improved by pruning individual trees.

(ii) A pruned tree may lead to enough reduction in prediction variance, such that developing a random forest with such pruned trees leads to negligible value addition (this can be shown by a similar exercise).

(7 points for code, 1 point for the precision-recall plot of the optimized random forest model)

[9]:
```
#Function to compute confusion matrix and prediction accuracy on test/train data
def confusion_matrix_data(data,actual_values,model,cutoff=0.5):
#Predict the values using the Logit model
    pred_values = model.predict_proba(data)[:,1]
# Specify the bins
    bins=np.array([0,cutoff,1])
#Confusion matrix
    cm = np.histogram2d(actual_values, pred_values, bins=bins)[0]
    cm_df = pd.DataFrame(cm)
    cm_df.columns = ['Predicted 0','Predicted 1']
    cm_df = cm_df.rename(index={0: 'Actual 0',1:'Actual 1'})
# Calculate the accuracy
    accuracy = 100*(cm[0,0]+cm[1,1])/cm.sum()
    fnr = 100*(cm[1,0])/(cm[1,0]+cm[1,1])
    precision = 100*(cm[1,1])/(cm[0,1]+cm[1,1])
    fpr = 100*(cm[0,1])/(cm[0,0]+cm[0,1])
    tpr = 100*(cm[1,1])/(cm[1,0]+cm[1,1])
```

```
    print("Accuracy = ", accuracy)
    print("Precision = ", precision)
    print("FNR = ", fnr)
    print("FPR = ", fpr)
    print("TPR or Recall = ", tpr)
    print("Confusion matrix = \n", cm_df)
    return (" ")
```

[10]:
```
train = pd.read_csv('train.csv')
test1 = pd.read_csv('test1.csv')
test2 = pd.read_csv('test2.csv')
train['ynum']=0
train.loc[train['y']=='yes','ynum']=1
test1['ynum']=0
test1.loc[test1['y']=='yes','ynum']=1
test2['ynum']=0
test2.loc[test2['y']=='yes','ynum']=1
train_dum = pd.concat([pd.get_dummies(train['education'],prefix='edu'),
                       pd.
 ↪get_dummies(train['month'],prefix='month'),train[['age','day']]],axis=1)
test1_dum = pd.concat([pd.get_dummies(test1['education'],prefix='edu'),
                       pd.
 ↪get_dummies(test1['month'],prefix='month'),test1[['age','day']]],axis=1)
test2_dum = pd.concat([pd.get_dummies(test2['education'],prefix='edu'),
                       pd.
 ↪get_dummies(test2['month'],prefix='month'),test2[['age','day']]],axis=1)
X = train_dum
Xtest1 = test1_dum
Xtest2 = test2_dum
y = train['ynum']
ytest1 = test1['ynum']
ytest2 = test2['ynum']
```

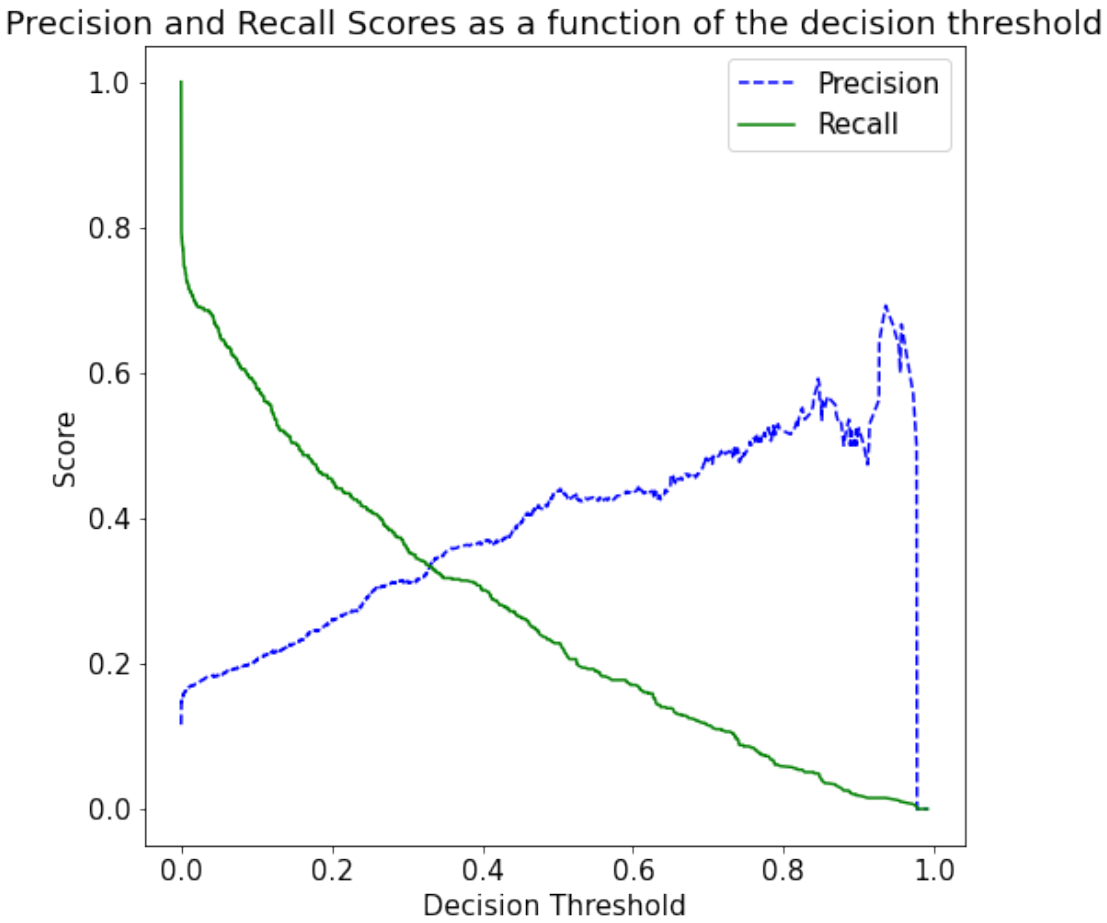
[11]:
```
start_time = time.time()

n_samples = train.shape[0]
n_features = train.shape[1]

params = {'n_estimators': [250,300,350],
          'max_features': range(1,19)}

param_list=list(it.product(*(params[Name] for Name in list(params.keys()))))

precision = [0]*len(param_list)
i=0
for pr in param_list:
```

```
    model =␣
 ↪RandomForestClassifier(random_state=1,oob_score=True,verbose=False,n_estimators␣
 ↪= pr[0],
                                    max_features=pr[1],n_jobs=-1).fit(X,y)
    oob_pred = model.oob_decision_function_[:,1]
    bins=np.array([0,0.5,1])
    cm = np.histogram2d(y, oob_pred, bins=bins)[0]
    precision[i] = 100*(cm[1,1])/(cm[0,1]+cm[1,1])
    i=i+1

end_time = time.time()
print("time taken = ", (end_time-start_time)/60, " minutes")
print("max precision = ", np.max(precision))
print("params= ", param_list[np.argmax(precision)])
```

```
time taken =  2.8190293510754905  minutes
max precision =  39.23994697304463
params=  (300, 1)
```

The optimal parameters for maximizing precision are 300 trees and 1 predictor being randomly selected to split each non-terminal node.

[12]:
```
#Precision vs recall curve based on the optimized model
model = RandomForestClassifier(n_estimators=300,␣
 ↪random_state=1,max_features=1,n_jobs=-1,oob_score=False).fit(X, y)
ypred = model.predict_proba(Xtest1)[:, 1]
p, r, thresholds = precision_recall_curve(ytest1, ypred)
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.figure(figsize=(8, 8))
    plt.title("Precision and Recall Scores as a function of the decision␣
 ↪threshold")
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.ylabel("Score")
    plt.xlabel("Decision Threshold")
    plt.legend(loc='best')

plot_precision_recall_vs_threshold(p, r, thresholds)
```



The tuned random forrest model fails to achieve the objective of having a precision and recall of less than 40%. However, a single pruned decision tree model could meet the objective.

Q3

We aim to classify stars as Galaxies or Quasars *(Quasars are extremely luminous active galactic nucleus, powered by a supermassive black hole)* based on their spectral characteristics. The data (reference) consists of observations of space taken by the SDSS (Sloan Digital Sky Survey). Every observation is described by 9 feature columns and 1 class column which identifies it to be either a galaxy or quasar:

`alpha` = Right Ascension angle (at J2000 epoch)
`delta` = Declination angle (at J2000 epoch)
`u` = Ultraviolet filter in the photometric system
`g` = Green filter in the photometric system
`r` = Red filter in the photometric system
`i` = Near Infrared filter in the photometric system
`z` = Infrared filter in the photometric system
`cam_col` = Camera column to identify the scanline within the run

`class` = object class (galaxy or quasar object)
`redshift` = redshift value based on the increase in wavelength

Develop and tune a random forest model using *Stellar_Classification_train.csv* to classify an observations as a Galaxy or Quasar. The model must have a ROC-AUC of at least 99% on both *Stellar_Classification_train.csv* and *Stellar_Classification_test.csv*. Print the ROC-AUC for both the datasets.

Hint: Consider 500 trees, and find the value of `max_features` that maximizes the out-of-bag (OOB) ROC-AUC.

(7 points for code, 1 point for answer)

[13]:
```
train = pd.read_csv('Stellar_Classification_train.csv')
test = pd.read_csv('Stellar_Classification_test.csv')
Xtrain = train.drop(columns = 'class')
Xtest = test.drop(columns = 'class')
ytrain = pd.get_dummies(train['class'])['QSO']
ytest = pd.get_dummies(test['class'])['QSO']
```

[14]:
```
oob_auc = {}
for i in range(1,10):
    model = RandomForestClassifier(n_estimators=500,␣
 ↪random_state=1,n_jobs=-1,max_features = i,
                                    #max_depth=16,max_leaf_nodes=44,
                                oob_score=True).fit(Xtrain, ytrain)
    oob_pred = model.oob_decision_function_[:,1]
    fpr, tpr, auc_thresholds = roc_curve(ytrain, oob_pred)
    oob_auc[i]=auc(fpr, tpr)
```

[15]:
```
#Optimal value of max_features is:
print("Optimal value of max_features is:", list(oob_auc.keys())[np.
 ↪argmax(list(oob_auc.values()))])
```

```
Optimal value of max_features is: 3
```

[16]:
```
model = RandomForestClassifier(n_estimators=500,␣
 ↪random_state=1,n_jobs=-1,max_features = 3,
                                oob_score=False).fit(Xtrain, ytrain)
ypred = model.predict_proba(Xtrain)[:,1]
fpr, tpr, auc_thresholds = roc_curve(ytrain, ypred)
print("ROC-AUC on train data = ",auc(fpr, tpr))
```

```
ROC-AUC on train data =  1.0
```

[17]:
```
model = RandomForestClassifier(n_estimators=500,␣
 ↪random_state=1,n_jobs=-1,max_features = 3,
                                oob_score=False).fit(Xtrain, ytrain)
ypred = model.predict_proba(Xtest)[:,1]
fpr, tpr, auc_thresholds = roc_curve(ytest, ypred)
```

```
print("ROC-AUC on test data = ",auc(fpr, tpr))
```

```
ROC-AUC on test data =  0.990037663113758
```

Q4

Can a non-linear monotonic transformation of predictors (such as *log()*, *sqrt()* etc.) be useful in improving the accuracy of decision tree models?

(3 points for answer)

No. A non-linear transformation of predictors will only change the values of the transformed predictors at which a node splits into child nodes. The reduction in Residual sum of squares (RSS) in case of regression or Gini index in case of classification will be the same as with untransformed predictors. As the criteria used to grow trees uses metrics (such as RSS or Gini index) that depend on the values of the response variable and not the predictors, a non-linear monotonic transformation of predictors will not make any difference to a decision tree model.