STAT 303-3 Assignment 2 Complete

April 21, 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, April 27th**. Good luck!

Submissions will be graded with a maximum of 55 points – 50 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 27.5/55 in the assigment, your score will be scaled to 50/100

1 Part 1

Q1a

Develop a decision tree model on the data *house_feature_train.csv* to predict *house_price*. Use all the predictors except *house_id*. What is the:
(i) model depth and,
(ii) number of leaves in the model?

Hint: Use the attribute *tree_* to get the attributes of the tree model. Example: *model.tree.n_leaves* is the number of leaves of the tree *model*.

(2 points for code, 1 point for answer)

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.metrics import␣
 ↪roc_curve,mean_squared_error,precision_recall_curve, auc, make_scorer,␣
 ↪recall_score, accuracy_score, precision_score, confusion_matrix
from sklearn.model_selection import KFold,StratifiedKFold,GridSearchCV,␣
 ↪ParameterGrid
from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
```

[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]:
```
#Defining the object to build a regression tree
model = DecisionTreeRegressor(random_state=1)

#Fitting the regression tree to the data
model.fit(X, y)

print("Model depth =", model.tree_.max_depth)
print("Number of leaves =", model.tree_.n_leaves)
```

```
Model depth = 16
Number of leaves = 249
```

Q1b

Find the RMSE of the model developed in the previous question on *house_feature_test.csv*.

(1 point for code, 1 point for answer)

[5]:
```
pred=(model.predict(Xtest))
np.sqrt(mean_squared_error(ytest, pred))
```

[5]: 440.11904192007887

Q1c

For the model developed in *Q1a*, use 5-fold cross validation to optimize the *depth, number of leaves, minimum number of observations in a node required to split the node*, and *minimum number of observations required in a leaf*. Report the optimal values of these parameters.

Use the following arguments:
(i) Use `random_state=1` when defining the `KFold` object,
(ii) Since you are using `random_state=1` while defining the `KFold` object, you will need to use the `shuffle = True` argument,
(iii) Use `random_state=1` with the `DecisionTreeRegressor` object.

Note: It is computationally expensive to consider a wide range of values of all the parameters. An optimization over any reasonable range (that your laptop can handle) is good enough. Different people may get different models, based on the range of values they consider to optimize the parameters.

Hint: The parameters of the `DecisionTreeRegressor` object to be optimized are `max_depth, max_leaf_nodes, min_samples_split and min_samples_leaf`.

(4 points for code, 1 point for answer)

[6]:
```
#Finding cross validation error for trees over a different parameter values.
parameters = {'max_depth':range(3,9),'max_leaf_nodes':
 ↪range(6,20),'min_samples_split':range(5,12),
              'min_samples_leaf':range(2,8)}
cv = KFold(n_splits = 5,shuffle=True,random_state=1)
```

```
model = GridSearchCV(DecisionTreeRegressor(random_state=1), parameters,␣
  ↪n_jobs=-1,verbose=1,cv=cv)
model.fit(X, y)
print (model.best_score_, model.best_params_)
```

```
Fitting 5 folds for each of 3528 candidates, totalling 17640 fits
0.6524861426250601 {'max_depth': 3, 'max_leaf_nodes': 8, 'min_samples_leaf': 7,
'min_samples_split': 5}
```

Q1d

Update the decision tree model in *Q1a* based on the optimal values of parameters found in *Q1c*. Find the RMSE of the updated model on *house_feature_test.csv*.

(2 points for code, 1 point for answer)

[7]:
```
model = DecisionTreeRegressor(random_state=1,␣
  ↪max_depth=3,max_leaf_nodes=8,min_samples_split=5,min_samples_leaf=7)
model.fit(X, y)
#RMSE on test data
print("RMSE =", np.sqrt(mean_squared_error(ytest, (model.predict(Xtest)))))
```

```
RMSE = 428.4516679633684
```

Q1e

A decision tree model has varying levels of prediction accuracy over different segments of data. What are the charecterisitics of the houses for which the decision tree model, developed in *Q1d*, provides the most accurate estimate of *house_price*? Follow the steps below to answer this question.

For the model developed in *Q1d*:
(i) Find the leaf that has the least *mean squared error (MSE)*
(ii) Find the decision rules that classify an observation to the leaf identified in (i)
(iii) Use the decision rules in (ii) to answer the question. For example, an answer can look like - "For houses with *house_age>3* and *number_convenience_stores<=2*, the model provides the most accurate estimate of *house_price*.

You may use the following functions to help you with the above steps. Given the functions, the problem is quite straightforward.

(3 points for code, 1 point for answer)

[8]:
```
#The following function finds indices of the leaf nodes, given the decision␣
  ↪tree model.
#The indices of nodes in a decision tree starts at the top from 0.
#The child nodes obtained from the first split have indices 1 (left) and 2␣
  ↪(right),
#the child nodes obtained from the second split have indices 3 (left) and 4␣
  ↪(right), and so on.
```

```python
def leaf_nodes_indices(model):
    children_left = model.tree_.children_left
    children_right = model.tree_.children_right
    leaf_nodes = []
    for i in range(model.tree_.node_count):
        if children_left[i] == children_right[i]:
            leaf_nodes.append(i)
    return leaf_nodes

#The following function finds the *mean squared error* of nodes with indices␣
 ↪*node_indices*
def mse(model,node_indices):
    return model.tree_.impurity[node_indices]

#The following function gives the decision rules for a node with index as␣
 ↪*node_index*
def decision_rules(model,node_index):
    child_node = node_index
    node_list=[]
    children_left = model.tree_.children_left
    children_right = model.tree_.children_right
    features=model.tree_.feature
    fnames = X.columns
    threshold = model.tree_.threshold
    p=1
    while p>0:
        if node_index%2>0:
            p= np.where(children_left==node_index)[0][0]
        else:
            p= np.where(children_right==node_index)[0][0]
        node_list.append(p)
        node_index=p
    node_list.reverse()
    node_list.append(child_node)
    cc=1
    for n in node_list[0:(len(node_list)-1)]:
        cnode = node_list[cc]
        if cnode%2==0:
            ineq_sign = ">"
        else:
            ineq_sign = "<="
        print("Split "+ str(cc)+":
 ↪"+fnames[features][n]+ineq_sign+str(threshold[n]))
        cc=cc+1
    node_list=[]
    return ""
```

```
[9]: leaves = leaf_nodes_indices(model)
rmse_leaves=mse(model,leaves)
min_rmse_leaf = leaves[np.argmin(rmse_leaves)]
decision_rules(model,min_rmse_leaf)
```

```
Split 1:distance_MRT>390.7689971923828
Split 2:latitude<=24.964895248413086
Split 3:latitude<=24.950759887695312
```

```
[9]: ''
```

For the houses with distance_MRT>390.76 and latitude<=24.95, the decision tree model provides the most accurate estimate.

Q1f

Predict the *house_price* of those houses in *house_feature_test.csv* that have the characteristics identified in *Q1e*. What is the RMSE?

(2 points for code, 1 point for answer)

```
[10]: Xtest_sample = Xtest.loc[(Xtest['distance_MRT']>390.7689971923828) &␣
↪(Xtest['latitude']<=24.950759887695312),:]
ytest_sample = ytest[list(Xtest_sample.index)]
print("RMSE =",np.sqrt(mean_squared_error(model.
↪predict(Xtest_sample),ytest_sample)))
```

```
RMSE = 181.00326025389404
```

Q1g

The decision tree model uses a greedy algorithm to fit the data. Thus it may choose an inferior split early on leading to an inferior model. Drop *longitude* from the set of predictors, and redo *Q1c* and *Q1d*. Report the RMSE of the model on *house_feature_test.csv*.

(4 points for code, 1 point for answer)

```
[11]: X.drop(columns = 'longitude', inplace = True)
Xtest.drop(columns = 'longitude', inplace = True)
```

```
/var/folders/j1/fd1k80s525v8bz9vgqc71l0w0000gn/T/ipykernel_6385/3551689077.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X.drop(columns = 'longitude', inplace = True)
/var/folders/j1/fd1k80s525v8bz9vgqc71l0w0000gn/T/ipykernel_6385/3551689077.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Xtest.drop(columns = 'longitude', inplace = True)
```

[12]:
```
#Finding cross validation error for trees ranging from a depth of 1 to 19.
parameters = {'max_depth':range(3,9),'max_leaf_nodes':
 ↪range(6,20),'min_samples_split':range(5,12),
              'min_samples_leaf':range(2,8)}
cv = KFold(n_splits = 5,shuffle=True,random_state=1)
model = GridSearchCV(DecisionTreeRegressor(random_state=1), parameters,␣
 ↪n_jobs=-1,verbose=1,cv=cv)
model.fit(X, y)
print (model.best_score_, model.best_params_)
```

```
Fitting 5 folds for each of 3528 candidates, totalling 17640 fits
0.6991620441000876 {'max_depth': 6, 'max_leaf_nodes': 14, 'min_samples_leaf': 3,
'min_samples_split': 5}
```

[13]:
```
model = DecisionTreeRegressor(random_state=1,␣
 ↪max_depth=6,max_leaf_nodes=14,min_samples_split=5,min_samples_leaf=3)

model.fit(X, y)
print("RMSE =", np.sqrt(mean_squared_error(ytest, (model.predict(Xtest)))))
```

```
RMSE = 373.8228967519727
```

Q1h

Use minimal cost-complexity pruning to develop the decision tree for predicting house price. Use all the predictors except *house_id*. Also optimize *min_samples_split* while finding the optimal cost complexity parameter, *ccp_alpha*. Find the RMSE of the model developed with the optimal parameters on *house_feature_test.csv*.

Note: Different people may obtain different models.

(3 points for code, 1 point for answer)

[14]:
```
X =␣
 ↪train[['house_age','distance_MRT','number_convenience_stores','latitude','longitude']]
Xtest =␣
 ↪test[['house_age','distance_MRT','number_convenience_stores','latitude','longitude']]
```

[15]:
```
model = DecisionTreeRegressor(random_state = 1)#model without any restrictions
path= model.cost_complexity_pruning_path(X,y)# Compute the pruning path during␣
 ↪Minimal Cost-Complexity Pruning.
```

[16]:
```
alphas=path['ccp_alphas']
len(alphas)
```

[16]: 211

[17]:
```
start_time = time.time()
cv = KFold(n_splits = 10,shuffle=True,random_state=1)
trees_search = GridSearchCV(DecisionTreeRegressor(random_state=1), param_grid =␣
 ↪{'ccp_alpha':alphas,
                                                                                ␣
 ↪ 'min_samples_split':range(5,15)},
                     scoring =␣
 ↪'neg_mean_squared_error',n_jobs=-1,verbose=1,cv=cv)
trees_search.fit(X, y)
print (trees_search.best_score_, trees_search.best_params_)
total_time = time.time()-start_time
```

```
Fitting 10 folds for each of 2110 candidates, totalling 21100 fits
-358042.66174721776 {'ccp_alpha': 9035.422678061843, 'min_samples_split': 6}
```

[18]:
```
tree = DecisionTreeRegressor(ccp_alpha=9035.
 ↪422678061845,random_state=1,min_samples_split=6)

tree.fit(X, y)
pred = tree.predict(Xtest)
np.sqrt(mean_squared_error(ytest, pred))
```

[18]: 386.8299625550846

Part 2

The data for this question is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls, where bank clients were called to subscribe for a term deposit.

There is one train data - *train.csv*, which you will use to develop a model. There are two test datasets - *test1.csv* and *test2.csv*, which you will use to test your model. Each dataset has the following attributes about the clients called in the marketing campaign:

1) *age*: Age of the client

2) *education*: Education level of the client

3) *day*: Day of the month the call is made

4) *month*: Month of the call

5) *y*: did the client subscribe to a term deposit?

6) *duration*: Call duration, in seconds. This attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also,

after the end of the call y is obviously known. Thus, this input should only be included for inference purposes and should be discarded if the intention is to have a realistic predictive model.

(Raw data source: Source. Do not use the raw data source for this assingment. It is just for reference.)

Note: 1. For optimizing models in this section you are free to choose any method you like - (i) cost complexity pruning (optimizing `ccp_alpha`) or (ii) optimizing other decision tree parameters (parameters other than `ccp_alpha`), or (iii) a combination of (i) and (ii).
2. Different people may get different models based on the parameters they consider for optimization. It is computationally infeasible to consider all possible parameter values.

2 Q2a

Develop a **decision tree model** (using *train.csv*) to predict the probability of a client subscribing to a term deposit based on *age, education, day* and *month.*The model must have both **precision and recall higher than 40%** on *train.csv*, *test1.csv* and *test2.csv*. Print the confusion matrices for all the three datasets - *train.csv*, *test1.csv* and *test2.csv*, along with their precision and recall.

Hints:
(i) Make dummy variables for categorical predictors
(ii) Optimize the depth and number of leaves of the decision tree,
(ii) Make a precision-recall curve to find the threshold probability satisfying the criteria.

(6 points for code, 1 point for answer)

[19]:
```
#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 (" ")
```

[20]:
```
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']
```

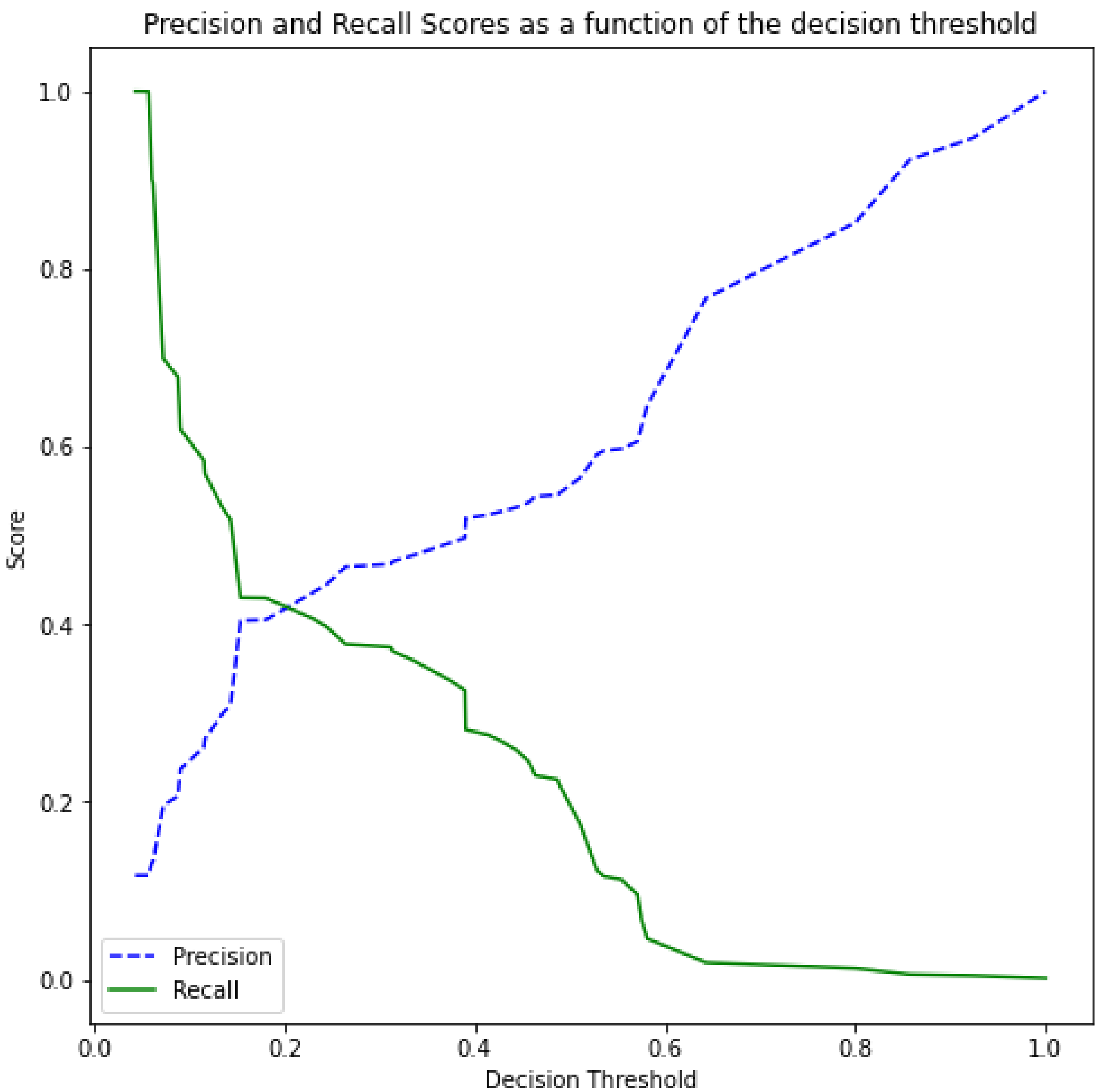
[21]:
```
#Defining parameters and the range of values over which to optimize
param_grid = {
    'max_depth': range(10,25),
    'max_leaf_nodes': range(30,50),
}
```

[22]:
```
#Grid search to optimize parameter values

skf = StratifiedKFold(n_splits=5)#The folds are made by preserving the␣
 ↪percentage of samples for each class.

#Minimizing FNR is equivalent to maximizing recall
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid,␣
 ↪scoring=['precision','recall',
                                                                              ␣
 ↪       'roc_auc'],
                           refit="recall", cv=skf, n_jobs=-1, verbose = True)
grid_search.fit(X, y)
```

```
print('Best params for precision')
print(grid_search.best_params_,grid_search.best_score_)
```

```
Fitting 5 folds for each of 300 candidates, totalling 1500 fits
Best params for precision
{'max_depth': 16, 'max_leaf_nodes': 44} 0.15911143459790264
```

[23]:
```
model = DecisionTreeClassifier(random_state=1, max_depth = 16,␣
 ↪max_leaf_nodes=44)
model.fit(X,y)
ypred = model.predict_proba(X)[:, 1]
p, r, thresholds = precision_recall_curve(y, 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)
```



From the above plot, a decision threshold probability of around 0.2 provides a precision and recall higher than 40%.

[24]:
```
model = DecisionTreeClassifier(random_state=1, max_depth = 16,␣
 ↪max_leaf_nodes=44)

model.fit(X,y)
cutoff=0.2
print(confusion_matrix_data(X,y,model,cutoff=cutoff))
print(confusion_matrix_data(Xtest1,ytest1,model,cutoff=cutoff))
print(confusion_matrix_data(Xtest2,ytest2,model,cutoff=cutoff))
```

```
Accuracy =  86.86285714285714
Precision =  43.55427974947808
```

```
FNR =  59.33235867446394
FPR =  7.000906266183325
TPR or Recall =  40.66764132553606
Confusion matrix =
            Predicted 0  Predicted 1
Actual 0      28733.0       2163.0
Actual 1       2435.0       1669.0

Accuracy =  86.82352941176471
Precision =  42.90780141843972
FNR =  59.12162162162162
FPR =  7.142857142857143
TPR or Recall =  40.87837837837838
Confusion matrix =
            Predicted 0  Predicted 1
Actual 0       4186.0        322.0
Actual 1        350.0        242.0

Accuracy =  86.44100958716494
Precision =  41.467576791808874
FNR =  59.02192242833052
FPR =  7.591854803010182
TPR or Recall =  40.97807757166948
Confusion matrix =
            Predicted 0  Predicted 1
Actual 0       4175.0        343.0
Actual 1        350.0        243.0
```

3 Q2b

Update the parameters of the model developed in *Q2a* (if necessary) to maximize the probaility that the model will predict a higher probability of response for a customer who signs up for the term deposit as compared to the customer who does not sign up, i.e., maximize the ROC-AUC. Report the ROC-AUC of the developed model on *train.csv*.

(3 points for code, 1 point for answer)

[25]:
```python
#Defining parameters and the range of values over which to optimize
param_grid = {
    'max_depth': range(15,25),
    'max_leaf_nodes': range(55,70),
}
```

[26]:
```python
#Grid search to optimize parameter values

skf = StratifiedKFold(n_splits=5)#The folds are made by preserving the␣
 ↪percentage of samples for each class.
```

```
#Minimizing FNR is equivalent to maximizing recall
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid,␣
 ↪scoring=['precision','recall',
                                                                              ␣
 ↪       'roc_auc'],
                             refit="roc_auc", cv=skf, n_jobs=-1, verbose = True)
grid_search.fit(X, y)

print('Best params for precision')
print(grid_search.best_params_,grid_search.best_score_)
```

```
Fitting 5 folds for each of 150 candidates, totalling 750 fits
Best params for precision
{'max_depth': 16, 'max_leaf_nodes': 57} 0.718693458114862
```

[27]:
```
model = DecisionTreeClassifier(random_state=1, max_depth = 16,␣
 ↪max_leaf_nodes=57)
model.fit(X,y)
ypred = model.predict_proba(X)[:, 1]
fpr, tpr, auc_thresholds = roc_curve(y, ypred)
print("ROC-AUC=", auc(fpr, tpr))# AUC of ROC
```

```
ROC-AUC= 0.7305253127703528
```

4 Q2c

Suppose that the model developed in *Q2b* will be used to predict the clients who will respond positively to the campaign. Only those clients who are predicted to respond positively will be called during the marketing campaign. Assume that:
(i) A profit of \$100 is associated with a client who responds positively to the campaign,
(i) A loss of \$10 is associated with a client who responds negatively to the campaign

Find the threshold probability of classification, such that the net profit is maximized. Use *train.csv*

(3 points for code, 1 point for answer)

[28]:
```
r = train['ynum'].mean()
#Net profit = Profit from clients responding positively - loss from clients␣
 ↪responding negatively
auc_thresholds[np.argmax((r*100*tpr)-(1-r)*10*fpr)]
```

[28]: 0.09096692111959287

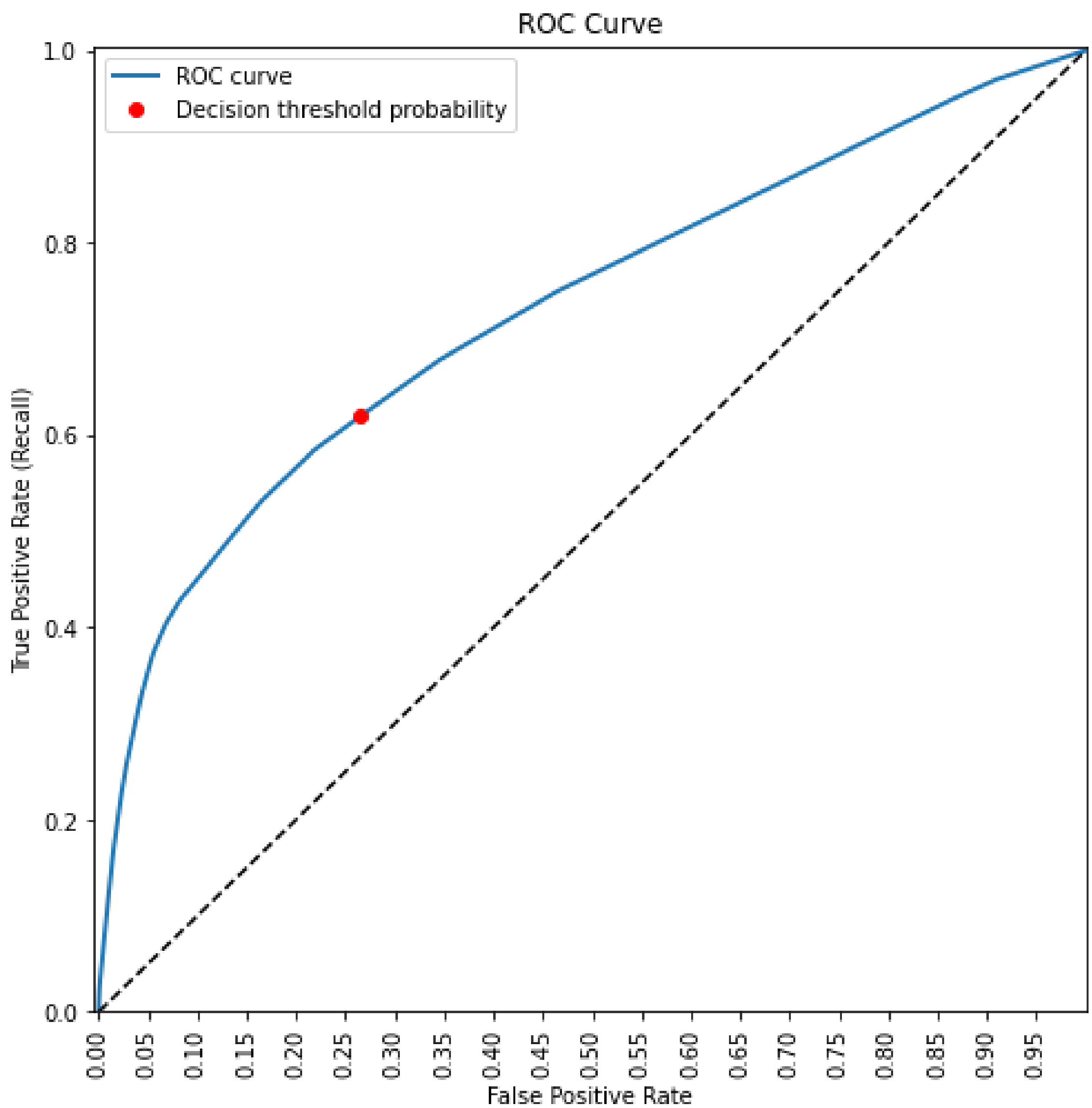
Threshold probability = 0.09

5 Q2d

Plot the ROC curve for the model developed in *Q2b*. Mark the point on the curve corresponding the threshold probability identified in *Q2c*.

(3 points for code)

[29]:
```
tprd=tpr[np.argmax((r*100*tpr)-(1-r)*10*fpr)]#TPR corresponding to the␣
↪threshold probability
fprd=fpr[np.argmax((r*100*tpr)-(1-r)*10*fpr)]#FPR corresponding to the␣
↪threshold probability

def plot_roc_curve(fpr, tpr, label=None):

    plt.figure(figsize=(8,8))
    plt.title('ROC Curve')
    plt.plot(fpr, tpr, linewidth=2, label="ROC curve")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.005, 1, 0, 1.005])
    plt.xticks(np.arange(0,1, 0.05), rotation=90)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate (Recall)")
    plt.plot(fprd,tprd,'o',color='red', label = "Decision threshold␣
↪probability")
    plt.legend()
plot_roc_curve(fpr, tpr)
```



6 Q2e

Among the months, which month is the most important in determining if a client will respond positively to the campaign, based on the model developed on *Q2b*.

(2 points for code, 1 point for answer)

[30]:
```
pd.concat([pd.Series(model.feature_importances_),pd.Series(X.columns)],axis=1).
↪sort_values(by = 0,ascending = False)
```

[30]:

	0	1
16	0.235016	age

17	0.189129	day
11	0.096281	month_mar
14	0.096272	month_oct
15	0.092751	month_sep
7	0.070700	month_feb
13	0.063421	month_nov
4	0.035806	month_apr
6	0.034656	month_dec
12	0.023313	month_may
8	0.021379	month_jan
0	0.010376	edu_primary
9	0.009406	month_jul
5	0.008407	month_aug
3	0.005579	edu_unknown
2	0.004065	edu_tertiary
10	0.003442	month_jun
1	0.000000	edu_secondary

The month of March is the most important.