IEOR 142A: Machine Learning and Data Analytics I, Fall 2024

Homework Assignment #3

Due October 23rd, 2024 at 11:59pm Pacific time

IMPORTANT NOTE: As indicated in the slides of Lecture 1 (pgs. 36-38) and the syllabus (pgs. 6-8), please list any resources outside of the course materials that you find helpful in completing the assignment (e.g. peers you discuss with, materials from different classes, blog posts, AI Tools, etc.). Please also be mindful of all policies in the syllabus concerning academic integrity and the use of AI Tools, including that you need to write your own solutions individually.

Problem 1: (20 points)

In this problem, we consider splitting when building a *regression tree* in the CART algorithm. We assume that there is a feature vector $X \in \mathbb{R}^p$ and dependent variable $Y \in \mathbb{R}$. We have collected a training dataset $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$ for all $i = 1, \ldots, n$. We also assume, for simplicity, that we are considering the initial split at the top (root node) of the tree. An arbitrary split simply divides the training dataset into a partition of size two. By appropriately reshuffling the data, we can represent this partition (again for simplicity) via two sub-datasets $(x_1, y_1), \ldots, (x_N, y_N)$ and $(x_{N+1}, y_{N+1}), \ldots, (x_n, y_n)$ where N is the index of the last observation included in the first set. Assume throughout that our impurity function is the RSS error – the standard choice for a regression tree.

Please answer the following:

a) (5 points) What is the total impurity value before the split? (This is the total impurity of the "null tree" or the "baseline model".)

b) (5 points) What is the total impurity value after the split? (This is the total impurity of the tree with the split as defined above.)

c) (10 points) Show that the total impurity value after the split is always less than or equal to the total impurity value before the split, i.e., splitting never increases the total impurity cost function. (Hint: you can use the fact that, given a sequence of real numbers $z_1, z_2, \ldots, z_n$, the mean $\bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i$ is the minimizer of the function $\text{RSS}(z) = \sum_{i=1}^{n}(z_i - z)^2$.)

a) impurity function = RSS function

$$\text{RSS}(z) = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ → mean of all dependent variable values

fmn total impurity before split = RSS across entire dataset

$$\text{RSS}_{\text{before split}} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

b) after split, we will have 2 datasets

subdataset #1 : $(x_1, y_1), \ldots, (x_M, y_M)$

subdataset #2 : $(x_{M+1}, y_{M+1}), \ldots, (x_n, y_n)$

for each subdataset, we calculate its own RSS with the means for each dataset to be used ($\bar{y}_1$ and $\bar{y}_2$)

total impurity after split = sum of the 2 subdata RSS

$$\Rightarrow \text{RSS after split} = \sum_{i=1}^{M} (y_i - \bar{y}_1)^2 + \sum_{i=M+1}^{n} (y_i - \bar{y}_2)^2$$

c) show : RSS after $\leq$ RSS before

what is significance of means?

↳ the mean used to minimize RSS

$\bar{y}$ mean minimizes RSS over entire dataset

$\bar{y}_1$ mean minimizes subdataset 1

$\bar{y}_2$ mean minimizes subdataset 2

calculating RSS for entire dataset using global mean $\bar{y}$ might not account for the specific structural differences between specific ranges of our data!

RSS allowing each subdataset to have its own means ($\bar{y}_1$ $\bar{y}_2$ respectively) will fit for the data in each subset $\Rightarrow$ likely reducing overall sum of squared residuals

Splitting dataset at any pt will not increase the RSS when being minimized at their mean:

thus RSS after $\leq$ RSS before

$\Rightarrow$ splitting allows more flexibility in fitting data by calculating more specific different means for different subsets

IEOR142AHW3

October 24, 2024

1 IEOR 142A HW 3

Joy Zhang

[48]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
import random
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, mean_absolute_error
from sklearn.model_selection import GridSearchCV
```

1.0.1 Problem 1

Work attached above!

1.0.2 Problem 2

[49]:
```
yelp_test = pd.read_csv("yelp_test.csv")
yelp_train = pd.read_csv("yelp_train.csv")
yelp_test.head()
```

[49]:

	stars	review_count	GoodForKids	Alcohol	\
0	2.5	98	TRUE	'none'	
1	4.5	78	TRUE	(Missing)	
2	4.0	316	TRUE	'none'	
3	4.5	641	TRUE	'beer_and_wine'	
4	3.0	179	TRUE	'beer_and_wine'	

```
  BusinessAcceptsCreditCards    WiFi BikeParking ByAppointmentOnly  \
0                       TRUE  'free'        TRUE         (Missing)
1                       TRUE  'free'       FALSE             FALSE
2                       TRUE  'free'        TRUE         (Missing)
3                       TRUE    'no'        TRUE             FALSE
4                       TRUE    'no'        TRUE         (Missing)

  WheelechairAccessible OutdoorSeating RestaurantsReservations DogsAllowed  \
0             (Missing)           TRUE                   FALSE   (Missing)
1                  TRUE      (Missing)               (Missing)        TRUE
2                  TRUE          FALSE                    TRUE       FALSE
3             (Missing)          FALSE                    TRUE       FALSE
4             (Missing)           TRUE                   FALSE   (Missing)

  Caters
0   TRUE
1   TRUE
2   TRUE
3  FALSE
4   TRUE
```

a) In general, there are several approaches for dealing with missing values in supervised learning. Each attribute with missing values in our dataset is a categorical feature and, in the subsequent models that you will build, you should treat (Missing) as an explicit category. Do you think this modeling approach is reasonable or not? Explain your answer. Treating missing values as an explicit category can be reasonable, especially for categorical features. In real life, the absence of information itself may carry meaning many times. For example, a cafe/restaurant/store that doesn't report that it provides WiFi or some specific service/item might be doing so for a reason, which could be indicative of its Yelp rating! Thus, including a Missing category allows our model to potentially capture this and not ignore the nuances.

bi) First build a linear regression model. Remember to use all of the provided independent variables, and you do not have to do variable selection in this problem. For each of the categorical variables, you should use (Missing) as the reference level to be incorporated into the intercept term.

[50]:
```
# Build linear regression model with missing as reference category for␣
 ↪categorical features
linear = smf.ols(formula="stars ~ review_count + C(GoodForKids,␣
 ↪Treatment(reference='(Missing)')) + \
                          C(Alcohol, Treatment(reference='(Missing)')) + \
                          C(BusinessAcceptsCreditCards,␣
 ↪Treatment(reference='(Missing)')) + \
                          C(WiFi, Treatment(reference='(Missing)')) + \
                          C(BikeParking, Treatment(reference='(Missing)')) + \
                          C(ByAppointmentOnly,␣
 ↪Treatment(reference='(Missing)')) + \
```

```
                            C(WheelechairAccessible,␣
 ↪Treatment(reference='(Missing)')) + \
                            C(OutdoorSeating, Treatment(reference='(Missing)'))␣
 ↪+ \
                            C(RestaurantsReservations,␣
 ↪Treatment(reference='(Missing)')) + \
                            C(DogsAllowed, Treatment(reference='(Missing)')) + \
                            C(Caters, Treatment(reference='(Missing)'))",␣
 ↪data=yelp_train).fit()

# Print model summary
print(linear.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  stars   R-squared:                       0.168
Model:                            OLS   Adj. R-squared:                  0.165
Method:                 Least Squares   F-statistic:                     50.60
Date:                Thu, 24 Oct 2024   Prob (F-statistic):          1.07e-227
Time:                        01:26:30   Log-Likelihood:                -7264.8
No. Observations:                6272   AIC:                         1.458e+04
Df Residuals:                    6246   BIC:                         1.476e+04
Df Model:                          25
Covariance Type:            nonrobust
==============================================================================
                                                                              
==============================================================
coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
--------------------------------------------------------------
Intercept
3.3655      0.040     83.427      0.000       3.286       3.445
C(GoodForKids, Treatment(reference='(Missing)'))[T.FALSE]
0.0026      0.045      0.057      0.955      -0.086       0.092
C(GoodForKids, Treatment(reference='(Missing)'))[T.TRUE]
-0.1003      0.035     -2.852      0.004      -0.169      -0.031
C(Alcohol, Treatment(reference='(Missing)'))[T.'beer_and_wine']
0.1960      0.047      4.197      0.000       0.104       0.288
C(Alcohol, Treatment(reference='(Missing)'))[T.'full_bar']
0.1078      0.043      2.489      0.013       0.023       0.193
C(Alcohol, Treatment(reference='(Missing)'))[T.'none']
0.0791      0.039      2.030      0.042       0.003       0.155
C(BusinessAcceptsCreditCards, Treatment(reference='(Missing)'))[T.FALSE]
0.4823      0.094      5.108      0.000       0.297       0.667
C(BusinessAcceptsCreditCards, Treatment(reference='(Missing)'))[T.TRUE]
0.0638      0.047      1.359      0.174      -0.028       0.156
C(WiFi, Treatment(reference='(Missing)'))[T.'free']
0.0727      0.035      2.103      0.036       0.005       0.140
```

```
C(WiFi, Treatment(reference='(Missing)'))[T.'no']
0.0833      0.033      2.495      0.013       0.018       0.149
C(WiFi, Treatment(reference='(Missing)'))[T.'paid']
-0.2748      0.114     -2.411      0.016      -0.498      -0.051
C(BikeParking, Treatment(reference='(Missing)'))[T.FALSE]
-0.1705      0.032     -5.313      0.000      -0.233      -0.108
C(BikeParking, Treatment(reference='(Missing)'))[T.TRUE]
-0.0930      0.029     -3.227      0.001      -0.149      -0.037
C(ByAppointmentOnly, Treatment(reference='(Missing)'))[T.FALSE]
0.1095      0.034      3.264      0.001       0.044       0.175
C(ByAppointmentOnly, Treatment(reference='(Missing)'))[T.TRUE]
0.2627      0.106      2.467      0.014       0.054       0.471
C(WheelechairAccessible, Treatment(reference='(Missing)'))[T.FALSE]
0.7830      0.089      8.801      0.000       0.609       0.957
C(WheelechairAccessible, Treatment(reference='(Missing)'))[T.TRUE]
0.4013      0.028     14.514      0.000       0.347       0.456
C(OutdoorSeating, Treatment(reference='(Missing)'))[T.FALSE]
-0.0548      0.040     -1.380      0.168      -0.133       0.023
C(OutdoorSeating, Treatment(reference='(Missing)'))[T.TRUE]
0.0365      0.042      0.862      0.389      -0.047       0.120
C(RestaurantsReservations, Treatment(reference='(Missing)'))[T.FALSE]
-0.2076      0.040     -5.135      0.000      -0.287      -0.128
C(RestaurantsReservations, Treatment(reference='(Missing)'))[T.TRUE]
-0.0185      0.045     -0.409      0.682      -0.107       0.070
C(DogsAllowed, Treatment(reference='(Missing)'))[T.FALSE]
0.2319      0.029      7.902      0.000       0.174       0.289
C(DogsAllowed, Treatment(reference='(Missing)'))[T.TRUE]
0.1467      0.056      2.639      0.008       0.038       0.256
C(Caters, Treatment(reference='(Missing)'))[T.FALSE]
-0.0921      0.030     -3.063      0.002      -0.151      -0.033
C(Caters, Treatment(reference='(Missing)'))[T.TRUE]
0.1300      0.033      3.987      0.000       0.066       0.194
review_count
0.0001   2.69e-05      4.037      0.000    5.59e-05       0.000
==============================================================================
Omnibus:                      153.279   Durbin-Watson:                   1.964
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              164.283
Skew:                          -0.396   Prob(JB):                     2.12e-36
Kurtosis:                       3.037   Cond. No.                     5.41e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.41e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The code above builds a linear regression model to predict the stars rating based on our various

independent variables, including categorical features with (Missing) as the reference level. This approach also assumes a linear relationship between the predictors and the response variable.

bii) Now build a regression tree model (using an implementation of the CART algorithm). Select the complexity parameter (i.e., ccp alpha in sklearn) value for the tree through cross-validation, and explain how you did the cross-validation and how you selected the complexity parameter value.

[51]:
```
# identify categorical columns that need encoding
categorical_cols = ['GoodForKids', 'Alcohol', 'BusinessAcceptsCreditCards',␣
 ↪'WiFi',
                    'BikeParking', 'ByAppointmentOnly', 'WheelechairAccessible',
                    'OutdoorSeating', 'RestaurantsReservations', 'DogsAllowed',␣
 ↪'Caters']

# using pd.get_dummies to perform OHE on categorical columns
X_train = pd.get_dummies(yelp_train.drop(['stars'], axis=1),␣
 ↪columns=categorical_cols)
y_train = yelp_train['stars']
X_test = pd.get_dummies(yelp_test.drop(['stars'], axis=1),␣
 ↪columns=categorical_cols)
y_test = yelp_test['stars']

# aligning columns of the training/test sets
X_train, X_test = X_train.align(X_test, join='left', axis=1, fill_value=0)

#Decision Tree Regressor
tree_model = DecisionTreeRegressor(random_state=42)
```

[52]:
```
# perform cross-validation to find optimal ccp_alpha: ATTEMPT 1
param_grid = {'ccp_alpha': [0.001, 0.50, 1, 10]}

grid_search = GridSearchCV(tree_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)


# fit the best tree
best_tree = grid_search.best_estimator_

print(f"Best ccp_alpha: {grid_search.best_params_['ccp_alpha']}")
```

```
Best ccp_alpha: 0.001
```

[53]:
```
# perform cross-validation to find optimal ccp_alpha: ATTEMPT 2
param_grid = {'ccp_alpha': [0.001, 0.0015, 0.002, 0.0025]}

grid_search = GridSearchCV(tree_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
# fit best tree
best_tree = grid_search.best_estimator_

print(f"Best ccp_alpha: {grid_search.best_params_['ccp_alpha']}")
```

```
Best ccp_alpha: 0.0015
```

The code above builds a regression tree model using scikit-learn to predict the stars ratings. This model splits the data into subsets based on feature values, allowing for complex relationships between predictors and the response variable without assuming a specific functional form.

To select the complexity parameter ccp_alpha for the decision tree, cross-validation is performed to evaluate the model's performance across multiple subsets of the training data. This process helps in finding the optimal balance between model complexity and predictive accuracy by systematically testing different values of ccp_alpha. In each iteration, the tree is built and pruned using a specific ccp_alpha, and its performance is assessed on validation data. The value of ccp_alpha that yields the best average performance across the folds is selected as the optimal parameter, ensuring that the model does not overfit to the training data while maintaining predictive power.

I first used a broader range of potential values [0.001, 0.50, 1, 10] to narrow it down to it being within [0.001, 0.0015, 0.002, 0.0025], and then finally concluded that the best ccp_alpha is around 0.0015.

biii) Using the test set data provided in the yelp test.csv file, compute the OSR2 values of your linear regression and regression tree models. Also, compute the MAE (mean absolute error) values of both models. How do you judge the performance of the two models?

[54]:
```
from sklearn.metrics import mean_absolute_error, r2_score

# Predictions on test set
linear_pred = linear.predict(yelp_test)
tree_pred = grid_search.predict(X_test)

# Compute OSR2 and MAE for both models
linear_mae = mean_absolute_error(yelp_test['stars'], linear_pred)
tree_mae = mean_absolute_error(y_test, tree_pred)

linear_osr2 = r2_score(yelp_test['stars'], linear_pred)
tree_osr2 = r2_score(y_test, tree_pred)

print("Linear Regression OSR2:", linear_osr2, "MAE:", linear_mae)
print("Regression Tree OSR2:", tree_osr2, "MAE:", tree_mae)
```

```
Linear Regression OSR2: 0.16250812470476927 MAE: 0.6301290815263458
Regression Tree OSR2: 0.14765568972236542 MAE: 0.6288947078941746
```

OSR² (Adjusted R-squared): - **Linear Regression**: 0.1625 - **Regression Tree**: 0.1477

The OSR² value indicates how well the model explains the variability of the response data around

its mean. An OSR^2 closer to 1 suggests a better fit, while a value closer to 0 suggests a poor fit. In this case, both models have relatively low OSR^2 values, with Linear Regression performing slightly better than the Regression Tree. However, neither model demonstrates a strong explanatory power for the response variable since both values are significantly below 0.5.

MAE (Mean Absolute Error): - **Linear Regression**: 0.6301 - **Regression Tree**: 0.6289

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. Lower values indicate better performance. Here, both models have similar MAE values, with the Regression Tree being marginally better. This suggests that both models have comparable prediction accuracy, with the Regression Tree being slightly more accurate in terms of absolute error.

- **OSR^2**: The Linear Regression model is slightly better at explaining variance than the Regression Tree, but both are low overall.
- **MAE**: The Regression Tree has a slightly lower MAE, indicating marginally better prediction accuracy.

ci) Construct a new variable in your training and test datasets called fourOrAbove. This variable should be equal to 1 if stars is greater than or equal to 4 and equal to 0 otherwise.

[55]:
```
# binary classify: 1 if stars above or equal to 4; 0 otherwise
yelp_train['fourOrAbove'] = (yelp_train['stars'] >= 4).astype(int)
yelp_test['fourOrAbove'] = (yelp_test['stars'] >= 4).astype(int)
```

d) Let us now work on building classification models for predicting fourOrAbove based on all of the provided features listed in Table 1. All of your models should, of course, be built only using the training data provided in the yelp train.csv file.

di) In this problem, we will weigh false positives and false negatives equally and therefore focus on accuracy (equivalently, error rate) as the primary performance metric. Do you think this modeling choice is reasonable or not? Explain your answer
Weighing false positives and false negatives equally has its pros and cons. I think it makes sense if the goal is to achieve a general level of accuracy across both classes and if the misclassification costs are balanced. However, in reality, sometimes the consequences of these misclassifications can differ:

A false negative (ex: failing to identify a high-quality review) may lead to a business losing potential customers, while a false positive (ex: overestimating a review) might mislead ratings but could be less damaging overall. Additionally, if the dataset is imbalanced, treating both types of errors equally could result in neglecting the performance of the minority class. A more balanced metric like precision/recall/F1 score could offer better insights into the model's ability to handle both high and low ratings fairly.

dii) A simple approach here is to use the previously built linear regression and regression tree models to address the classification task by thresholding their predictions at the value of 4. Write code for implementing this thresholding procedure for converting the predictions of your previously built regression models to predictions of fourOrAbove.

[56]:
```
# make predictions using the linear regression model
y_pred_lr_test = linear.predict(yelp_test)

# make predictions using the regression tree model
y_pred_tree_test = grid_search.predict(X_test)

# threshold the predictions for the linear regression model (4 and above)
y_pred_lr_class = (linear_pred >= 4).astype(int)

# threshold the predictions for the regression tree model
y_pred_tree_class = (tree_pred >= 4).astype(int)

print("Thresholded predictions for Linear Regression at value 4:",␣
 ↪y_pred_lr_class)
print("Thresholded predictions for Regression Tree at value 4:",␣
 ↪y_pred_tree_class)
```

```
Thresholded predictions for Linear Regression at value 4: 0       0
1       1
2       1
3       0
4       0
       ..
2683    0
2684    0
2685    1
2686    0
2687    0
Length: 2688, dtype: int64
Thresholded predictions for Regression Tree at value 4: [0 1 0 … 0 0 0]
```

diii) Now build a logistic regression model. Remember to use all of the provided independent variables, and you do not have to do variable selection in this problem. Please again use (Missing) as the reference level to be incorporated into the intercept term.

[58]:
```
# Build the logistic regression model using the binary outcome
logistic_model = smf.logit(formula="fourOrAbove ~ review_count + \
                        C(GoodForKids, Treatment(reference='(Missing)')) + \
                        C(Alcohol, Treatment(reference='(Missing)')) + \
                        C(BusinessAcceptsCreditCards,␣
 ↪Treatment(reference='(Missing)')) + \
                        C(WiFi, Treatment(reference='(Missing)')) + \
                        C(BikeParking, Treatment(reference='(Missing)')) + \
                        C(ByAppointmentOnly, Treatment(reference='(Missing)'))␣
 ↪+ \
                        C(WheelechairAccessible,␣
 ↪Treatment(reference='(Missing)')) + \
```

```
                        C(OutdoorSeating, Treatment(reference='(Missing)')) + \
                        C(RestaurantsReservations,␣
 ↪Treatment(reference='(Missing)')) + \
                        C(DogsAllowed, Treatment(reference='(Missing)')) + \
                        C(Caters, Treatment(reference='(Missing)'))",
                        data=yelp_train).fit()


print(logistic_model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.607102
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:            fourOrAbove   No. Observations:                 6272
Model:                          Logit   Df Residuals:                     6246
Method:                           MLE   Df Model:                           25
Date:                Thu, 24 Oct 2024   Pseudo R-squ.:                 0.1143
Time:                        01:26:33   Log-Likelihood:                -3807.7
converged:                       True   LL-Null:                       -4299.3
Covariance Type:            nonrobust   LLR p-value:                2.243e-191
==============================================================================
=============================================================
coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
-------------------------------------------------------------
Intercept
-0.4601      0.108     -4.248      0.000      -0.672      -0.248
C(GoodForKids, Treatment(reference='(Missing)'))[T.FALSE]
0.2015      0.128      1.576      0.115      -0.049       0.452
C(GoodForKids, Treatment(reference='(Missing)'))[T.TRUE]
-0.2144      0.099     -2.164      0.030      -0.409      -0.020
C(Alcohol, Treatment(reference='(Missing)'))[T.'beer_and_wine']
0.2021      0.134      1.514      0.130      -0.060       0.464
C(Alcohol, Treatment(reference='(Missing)'))[T.'full_bar']
-0.0376      0.124     -0.303      0.762      -0.281       0.206
C(Alcohol, Treatment(reference='(Missing)'))[T.'none']
0.3491      0.112      3.129      0.002       0.130       0.568
C(BusinessAcceptsCreditCards, Treatment(reference='(Missing)'))[T.FALSE]
1.2297      0.268      4.592      0.000       0.705       1.755
C(BusinessAcceptsCreditCards, Treatment(reference='(Missing)'))[T.TRUE]
0.0811      0.127      0.640      0.522      -0.167       0.330
C(WiFi, Treatment(reference='(Missing)'))[T.'free']
0.2002      0.097      2.056      0.040       0.009       0.391
C(WiFi, Treatment(reference='(Missing)'))[T.'no']
0.1415      0.094      1.499      0.134      -0.044       0.327
```

```
C(WiFi, Treatment(reference='(Missing)'))[T.'paid']
-1.0509      0.404     -2.603      0.009      -1.842      -0.260
C(BikeParking, Treatment(reference='(Missing)'))[T.FALSE]
-0.4720      0.092     -5.123      0.000      -0.653      -0.291
C(BikeParking, Treatment(reference='(Missing)'))[T.TRUE]
-0.2977      0.081     -3.679      0.000      -0.456      -0.139
C(ByAppointmentOnly, Treatment(reference='(Missing)'))[T.FALSE]
0.4313      0.096      4.474      0.000       0.242       0.620
C(ByAppointmentOnly, Treatment(reference='(Missing)'))[T.TRUE]
0.5197      0.317      1.638      0.101      -0.102       1.141
C(WheelechairAccessible, Treatment(reference='(Missing)'))[T.FALSE]
2.0195      0.316      6.383      0.000       1.399       2.640
C(WheelechairAccessible, Treatment(reference='(Missing)'))[T.TRUE]
1.0400      0.076     13.659      0.000       0.891       1.189
C(OutdoorSeating, Treatment(reference='(Missing)'))[T.FALSE]
-0.1877      0.111     -1.685      0.092      -0.406       0.031
C(OutdoorSeating, Treatment(reference='(Missing)'))[T.TRUE]
-0.0333      0.119     -0.281      0.779      -0.266       0.199
C(RestaurantsReservations, Treatment(reference='(Missing)'))[T.FALSE]
-0.4660      0.114     -4.082      0.000      -0.690      -0.242
C(RestaurantsReservations, Treatment(reference='(Missing)'))[T.TRUE]
-0.0524      0.128     -0.410      0.682      -0.303       0.198
C(DogsAllowed, Treatment(reference='(Missing)'))[T.FALSE]
0.6381      0.082      7.783      0.000       0.477       0.799
C(DogsAllowed, Treatment(reference='(Missing)'))[T.TRUE]
0.3871      0.158      2.443      0.015       0.077       0.698
C(Caters, Treatment(reference='(Missing)'))[T.FALSE]
-0.2490      0.086     -2.911      0.004      -0.417      -0.081
C(Caters, Treatment(reference='(Missing)'))[T.TRUE]
0.1515      0.092      1.653      0.098      -0.028       0.331
review_count
0.0006      0.000      5.561      0.000       0.000       0.001
==============================================================================
==========================================================
```

Above is the code for building a logistic regression model that classifies 'stars' ratings into a binary outcome (1 for ratings of 4 or 5, and 0 for ratings of 1, 2, or 3). This transformation is necessary because logistic regression requires a binary dependent variable!

The logistic regression analysis predicting the binary outcome variable indicates that approximately 11.43% of the variance in ratings is explained by the model, as reflected by the pseudo-R-squared value of 0.1143.

The log-likelihood value of -3807.7 suggests a reasonable model fit, with the likelihood ratio p-value of 2.243e-191 indicating that at least one predictor is statistically significant.

div) Now build a classification tree model (using an implementation of the CART algorithm). Select the complexity parameter (i.e., ccp alpha in sklearn) value for the tree through cross-validation, and explain how you did the cross-validation and

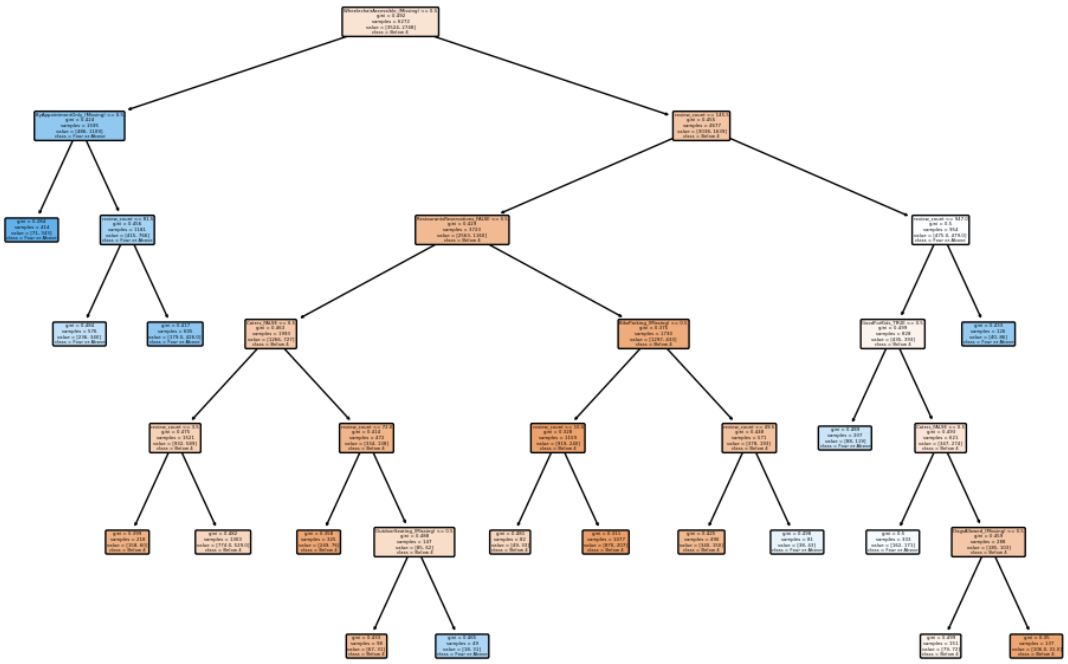
how you selected the complexity parameter value. Produce a diagram to show your decision tree.

[59]:
```
# identify categorical columns
categorical_cols = ['GoodForKids', 'Alcohol', 'BusinessAcceptsCreditCards',␣
 ↪'WiFi',
                    'BikeParking', 'ByAppointmentOnly', 'WheelechairAccessible',
                    'OutdoorSeating', 'RestaurantsReservations', 'DogsAllowed',␣
 ↪'Caters']

# OHE
X_train = pd.get_dummies(yelp_train.drop(['stars', 'fourOrAbove'], axis=1),␣
 ↪columns=categorical_cols)
y_train = yelp_train['fourOrAbove']
X_test = pd.get_dummies(yelp_test.drop(['stars', 'fourOrAbove'], axis=1),␣
 ↪columns=categorical_cols)
y_test = yelp_test['fourOrAbove']

X_train, X_test = X_train.align(X_test, join='left', axis=1, fill_value=0)

# DecisionTreeClassifier
clf_tree = DecisionTreeClassifier(random_state=42)

# perform grid search cross-validation to find the optimal ccp_alpha
grid_search = GridSearchCV(clf_tree, param_grid, cv=5, scoring='accuracy',␣
 ↪verbose=1)
grid_search.fit(X_train, y_train)

#fit the best tree (with optimal ccp_alpha)
best_tree_clf = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

[60]:
```
print(f"Best ccp_alpha: {grid_search.best_params_['ccp_alpha']}")

#graphing the tree
plt.figure(figsize=(12, 8))
plot_tree(best_tree_clf, filled=True, feature_names=X_train.columns,␣
 ↪class_names=['Below 4', 'Four or Above'], rounded=True)
plt.show()
```

```
Best ccp_alpha: 0.001
```



dv) Produce a table for evaluating the performance of the models that you have built. The table should consider the following models: a baseline model that predicts the most frequent outcome for fourOrAbove, the two regression models with thresholding, and all of the classification models built in this part of the problem. Additionally, the table should report the following performance metrics: accuracy (our primary performance metric), TPR, and FPR. All of these performance metrics should be computed using the test set data provided in the yelp test.csv file. Do the results seem reasonable to you? How do you judge the performance of the models and the trade-offs between different models? Which model would you recommend for this problem and why?

[61]:
```
from sklearn.metrics import confusion_matrix, accuracy_score


def compute_tpr_fpr(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    TN, FP, FN, TP = cm.ravel()
    TPR = TP / (TP + FN)  # True Positive Rate (Recall)
    FPR = FP / (FP + TN)  # False Positive Rate
    return TPR, FPR


most_frequent_class = y_train.value_counts().idxmax()
```

```
y_pred_baseline = np.full(y_test.shape, most_frequent_class)


y_pred_lr_class = (y_pred_lr_test >= 4).astype(int)


y_pred_tree_class = (y_pred_tree_test >= 4).astype(int)


y_pred_logit = logistic_model.predict(yelp_test)
y_pred_logit_class = (y_pred_logit >= 0.5).astype(int)


y_pred_tree_clf = best_tree_clf.predict(X_test)
```

[62]:
```
# Computing: Accuracy, TPR, and FPR for each
models = ['Baseline', 'Linear Regression (Thresholding)', 'Regression Tree␣
 ↪(Thresholding)', 'Logistic Regression', 'Classification Tree (CART)']
y_preds = [y_pred_baseline, y_pred_lr_class, y_pred_tree_class,␣
 ↪y_pred_logit_class, y_pred_tree_clf]

metrics = []
for model_name, y_pred in zip(models, y_preds):
    accuracy = accuracy_score(y_test, y_pred)
    tpr, fpr = compute_tpr_fpr(y_test, y_pred)
    metrics.append([model_name, accuracy, tpr, fpr])

# create a DataFrame to display the metrics
metrics_df = pd.DataFrame(metrics, columns=['Model', 'Accuracy', 'TPR', 'FPR'])

print(metrics_df)
```

```
                            Model  Accuracy       TPR       FPR
0                        Baseline  0.551339  0.000000  0.000000
1  Linear Regression (Thresholding)  0.616071  0.203980  0.048583
2    Regression Tree (Thresholding)  0.598214  0.129353  0.020243
3               Logistic Regression  0.665551  0.482587  0.185560
4       Classification Tree (CART)  0.639881  0.517413  0.260459
```

Logistic Regression has the highest accuracy (0.66) and a relatively high TPR (0.48), meaning it correctly identifies a good portion of positive cases. However, this comes with a moderate FPR (0.19), indicating that it occasionally predicts positive cases when they are actually negative. This model strikes a good balance between identifying positive cases and limiting false positives. **CART**, on the other hand, has the highest TPR (0.52), meaning it captures the most true positives, but also has the highest FPR (0.26), meaning it's more prone to false positives. While this model may be the best for maximizing positive predictions, it does so at the cost of more incorrect positive classifications.

In contrast, **Linear Regression (Thresholding)** and **Regression Tree (Thresholding)** are more conservative, with lower TPRs (0.20 and 0.13, respectively) but also very low FPRs (0.05 and 0.02). These models are better at avoiding false positives but are less effective at identifying true positives.

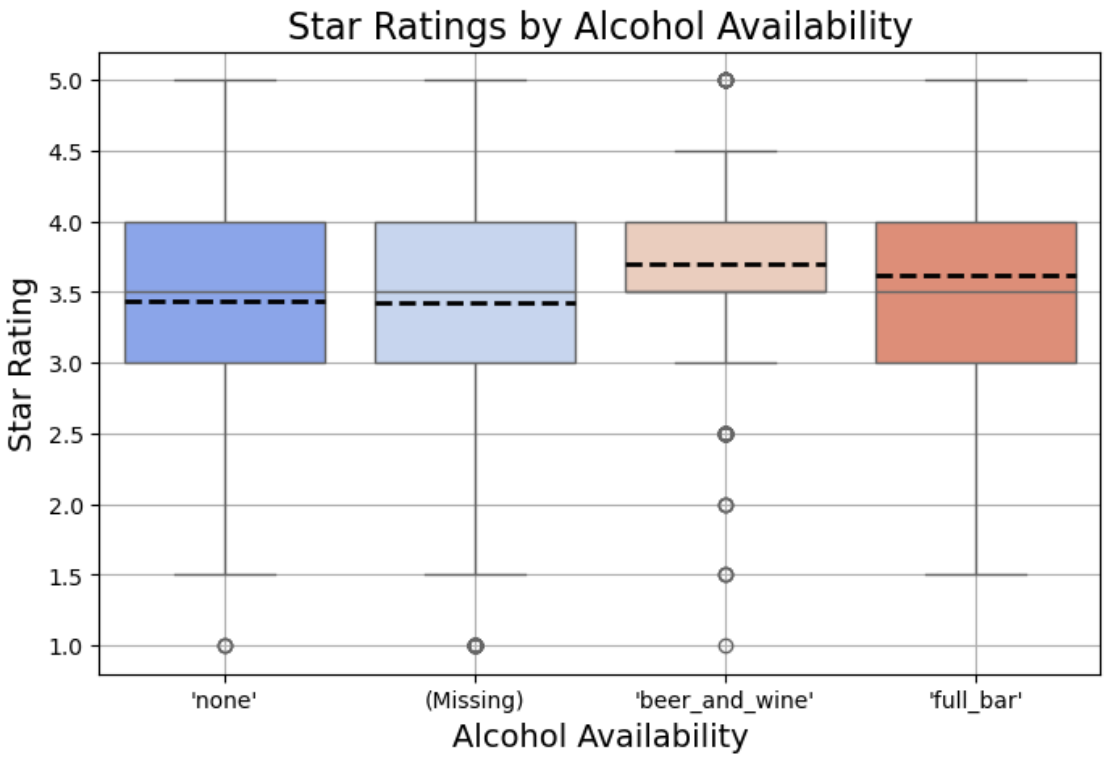
Given these results, **Logistic Regression** would generally be the best choice as it provides the best balance of accuracy and positive case identification, without being overly prone to false positives. However, if the goal of the problem is to capture as many positive cases as possible (with less concern for false positives), the **CART** model might be a better option.

e) Suppose that you are a data analytics professional working for Yelp and you have been tasked with producing a "how to guide" for Las Vegas restaurants, which is supposed to include tips for actions that restaurants may take to achieve a high star rating. Use the provided data to construct a list of three such tips.

1. Offer Alcohol to Boost Star Ratings Tip: Restaurants that offer beer and wine or have a full bar typically receive higher average star ratings compared to those that do not serve alcohol at all. The social aspect of alcohol, paired with a good meal, contributes to an improved customer experience, resulting in higher satisfaction and more positive reviews. Especially in Las Vegas, I am sure a lot of the demographic who stays there will want a place where they can drink and enjoy their time. Not to mention, many of these indulgers are relatively rich or they are tourists who are looking for a good time- having alcohol will likely increase their positive experiences and happiness levels to be inclined to leave a higher review.

[63]:
```
# Boxplot of star ratings by alcohol availability
plt.figure(figsize=(8, 5))
sns.boxplot(data=yelp_train, x='Alcohol', y='stars', palette='coolwarm',␣
 ↪showmeans=True,
            meanline=True, meanprops={"color": "black", "linewidth": 2,␣
 ↪"linestyle": "--"})

# Title and labels
plt.title('Star Ratings by Alcohol Availability', fontsize=16)
plt.xlabel('Alcohol Availability', fontsize=14)
plt.ylabel('Star Rating', fontsize=14)

# Show the plot
plt.grid(True)
plt.show()
```

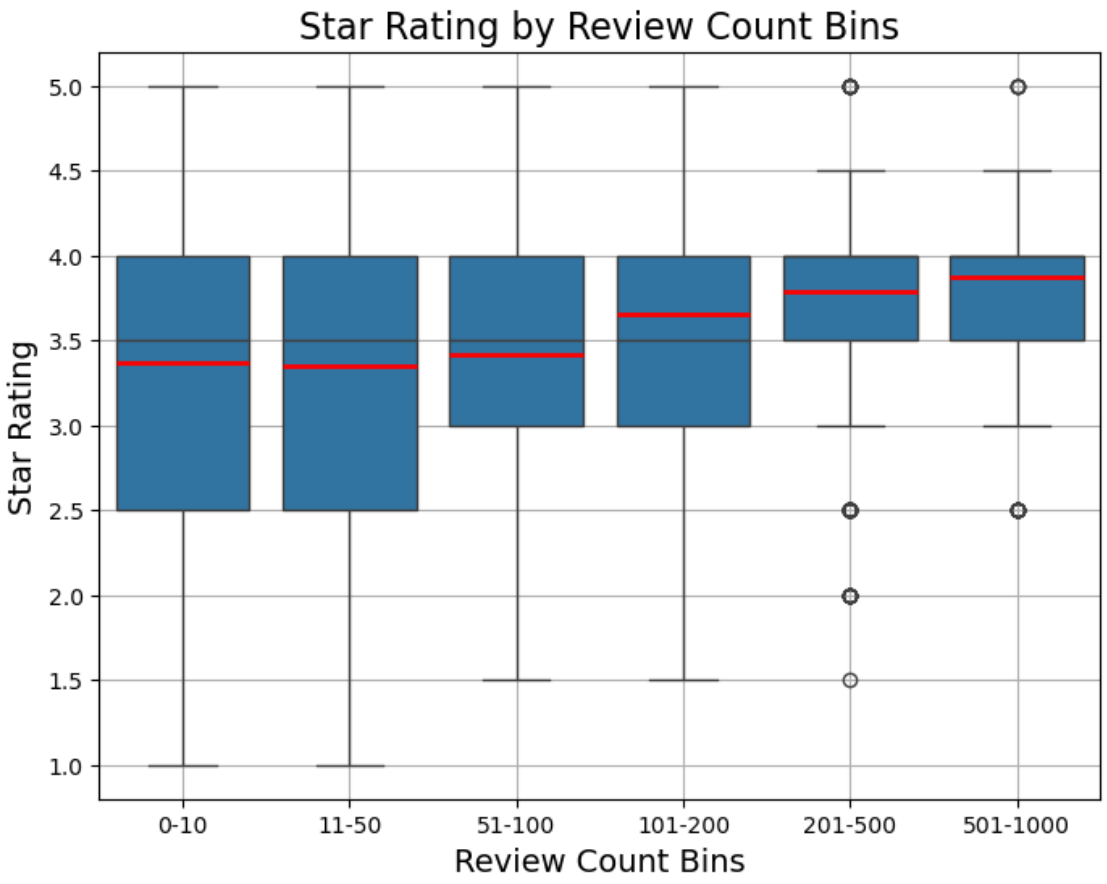
```
/tmp/ipykernel_86/3016845621.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=yelp_train, x='Alcohol', y='stars', palette='coolwarm',
showmeans=True,
```



2. Increase the Number of Reviews Through Engagement Tip: Actively ask customers to leave reviews, especially after a positive dining experience. The "review_count" feature is a critical indicator of a restaurant's popularity and credibility. When restaurants prompt satisfied customers to leave reviews, it helps improve this count and can mitigate the impact of any negative feedback. More reviews are left often because customers had positive experiences and felt compelled to share them. Thus, encouraging satisfied customers to leave reviews by offering a loyalty program, providing discounts for future visits, or just reminding them to leave feedback after a great experience would be helpful! Also, people in general trust reviews heavily and really care if a restaurant has a lot of reviews or not. Personally, that is always something I check before I decide: Do they have relatively high ratings? How many reviews do they have?

[64]:
```
# Create bins for review counts
yelp_train['review_bins'] = pd.cut(yelp_train['review_count'], bins=[0, 10, 50,␣
↪100, 200, 500, 1000],
                                   labels=['0-10', '11-50', '51-100',␣
↪'101-200', '201-500', '501-1000'])

# Boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(x='review_bins', y='stars', data=yelp_train, showmeans=True,␣
↪meanline=True,
            meanprops={"color": "red", "linewidth": 2, "linestyle": "-"})
```

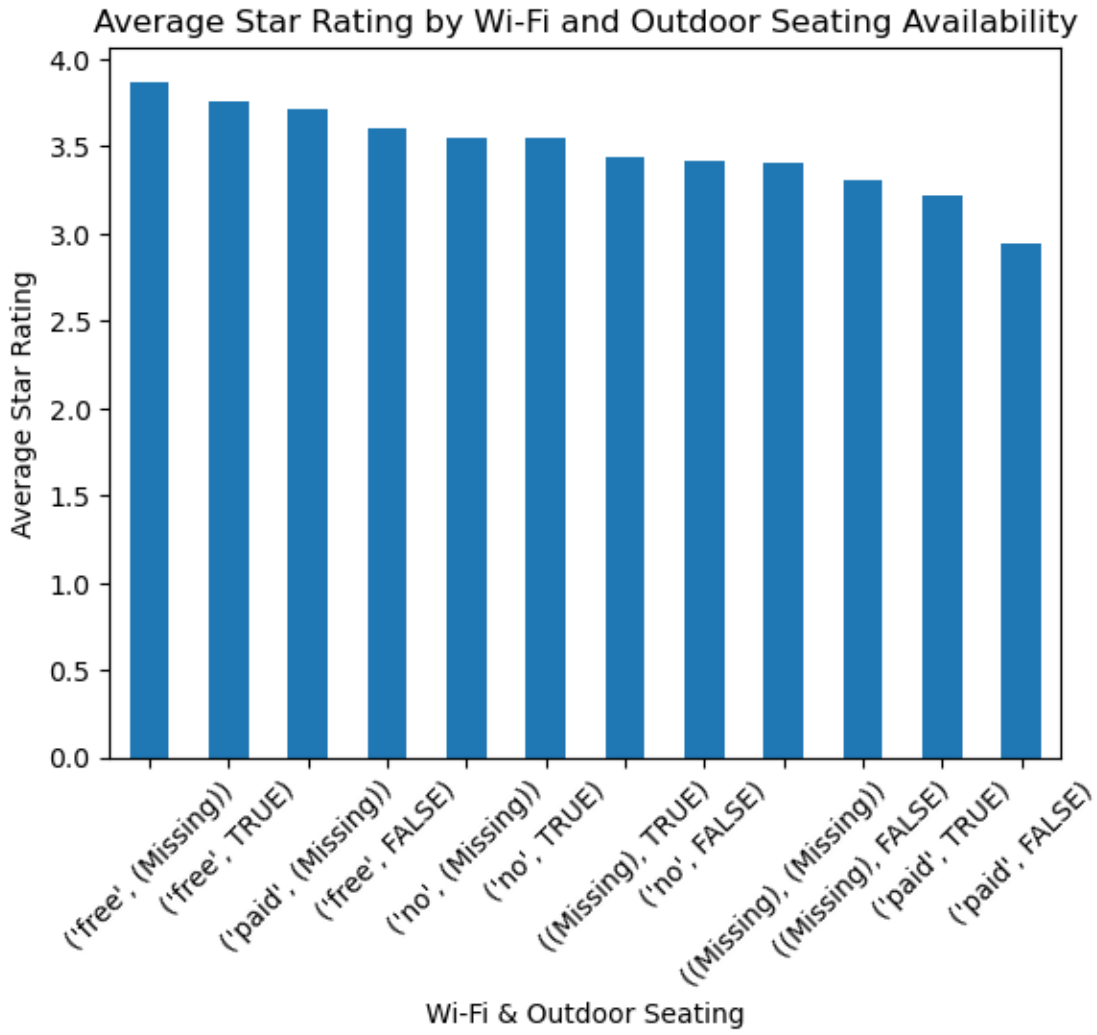
```
plt.title('Star Rating by Review Count Bins', fontsize=16)
plt.xlabel('Review Count Bins', fontsize=14)
plt.ylabel('Star Rating', fontsize=14)


plt.grid(True)
plt.show()
```



3. Enhance Customer Amenities Tip: Offer amenities like outdoor seating, Wi-Fi, and bike parking to cater to diverse customer needs. Restaurants with outdoor seating often attract more customers, especially in pleasant weather, contributing to a higher review count and potentially better ratings. Having wifi also allows customers to stay longer and prolong their enjoyable time. Bike parking allows on-the-go people/active people to also have a place to park their bikes, thus increasing the customer reach! From the bar graph below, you can see that having free Wifi and outdoor seating had an average higher than other options.

[65]:

```
# visualization: Bar plot showing the effect of Wi-Fi and Outdoor Seating on␣
 ↪star ratings
mean_ratings = yelp_train.groupby(['WiFi', 'OutdoorSeating'])['stars'].mean()
mean_ratings = mean_ratings.sort_values(ascending=False)
mean_ratings.plot(kind='bar')
plt.title('Average Star Rating by Wi-Fi and Outdoor Seating Availability')
plt.xlabel('Wi-Fi & Outdoor Seating')
plt.ylabel('Average Star Rating')
plt.xticks(rotation=45)
plt.show()
```



[ ]: