A20442087
Tiffany Wong

Assignment 3

CS484 Spring 2021
March 7, 2021

Question 1

- Answer

The misclassification rate of this decision tree diagram is $\frac{759 + 851}{8307} = 0.1938124473$.

- Explanation

The question states that an observation is misclassified if the predicted target category is different from the observed target category, meaning I have to look at the terminal nodes that aren't of the observed target values. The two nodes I'll be concentrating on is the terminal node of Trip_Time_journey <= 4534.5, where the entropy is 0.983, and Trip_Distance <= 170.682, where entropy is 0.75. When the entropy is not 0, it means that there are values that aren't perfectly classified in the sample, so I have to look at the number in the values array in each terminal node where the entropy is not 0. For the terminal node where entropy = 0.983, it's saying that there are 759 values in class 0 when it should be in class 1 and for the terminal node where entropy = 0.75, it's saying that there are 851 values in class 1 when it should be in class 0. To find the misclassification rate, I add up the number of values that are falsely predicted by their target category and divide by the total number of samples, which is 8307.

Question 2a

- Answer

The entropy value for the root node is 0.9489621493401781.

- Code

```
#%% question 2a

import pandas as pd
import numpy as np

# find the entropy value of the root node
# entropy = summation of (- (1/k)*log(1/k))
# root node is CAR_USE
# the two categories are commerical and private

df = pd.read_csv('/Users/tiffwong/Desktop/cs484/assignments/assignment 3/claim_history.csv')

car_use = df['CAR_USE']

commercial = 0
private = 0
total = len(car_use)

for i in range(total):
    if car_use[i] == 'Commercial':
        commercial += 1
    else:
        private += 1

entropy = (-commercial/total * (np.log2(commercial/total))) +
           (-private/total * (np.log2(private/total)))

print(entropy)
```

- Explanation

To find the entropy value of the root node, I had to find the count of 'commercial' and 'private' in the car_use column of the claim_history.csv. Then I used the equation for entropy, which is $\Sigma - p_i * (log_2(p_i))$. I found the number of times 'commercial' appears in the car_use column, divided by the total number of values in car_use, and then did the same to the rows of 'private'. I was able to find the entropy by adding those two values up.

Entropy(CAR_USE) = Entropy(3789, 6513)
= Entropy(0.3677926616, 0.6322073384)
= (-0.3677926616*$\log_2$(0.3677926616)) + (-0.6322073384*$\log_2$(0.6322073384))
= 0.9489621493401781

Question 2b

- Answer

CAR_TYPE

- Optimal split for CAR_TYPE is: ('Minivan', 'SUV', 'Sports Car')
- entropy = 0.7573352263531922
- values = [4543, 2668]

OCCUPATION

- Optimal split for OCCUPATION is: ('Blue Collar', 'Unknown', 'Student')
- entropy = 0.7148805225259208
- values = [2481 , 4730]

EDUCATION

- Optimal split for EDUCATION is: ('Below High School')
- entropy = 0.9343298080392602
- values = [6154 , 1057]

- Code

```
#%% entropy interval split function

def EntropyIntervalSplit (
   inData,          # input data frame (predictor in column 0 and target in column 1)
   split):          # split value

   dataTable = inData
   dataTable['LE_Split'] = False

   for k in dataTable.index:
       if dataTable.iloc[:,0][k] in split:
           dataTable['LE_Split'][k] = True

   crossTable = pd.crosstab(index = dataTable['LE_Split'],
                            columns = dataTable.iloc[:,1],
                            margins = True, dropna = True)

   nRows = crossTable.shape[0]
   nColumns = crossTable.shape[1]

   tableEntropy = 0

   for iRow in range(nRows-1):
      rowEntropy = 0

      for iColumn in range(nColumns):
         proportion = (crossTable.iloc[iRow,iColumn] /
                       crossTable.iloc[iRow, (nColumns-1)])

         if (proportion > 0):
            rowEntropy -= proportion * np.log2(proportion)

      tableEntropy += rowEntropy *  crossTable.iloc[iRow, (nColumns-1)]

   tableEntropy = tableEntropy /  crossTable.iloc[(nRows-1), (nColumns-1)]

   return(tableEntropy)
```

```
#%% minimum entropy function

def calculate_min_entropy(df, variable, combinations):
    inData1 = df[[variable, "Labels"]]
    all_entropy = []

    for i in combinations:
        buf = EntropyIntervalSplit(inData1, list(i))
        all_entropy.append((buf, i))

    return min(all_entropy)
```

```
#%% find best split for occupation

entropy_occupation = calculate_min_entropy(features_train,
                                           "OCCUPATION",
                                           occupation_combinations)

print(entropy_occupation)


#%% best split for car type

entropy_cartype = calculate_min_entropy(features_train,
                                        "CAR_TYPE",
                                        car_type_combinations)

print(entropy_cartype)


#%% best split for education

entropy_education = calculate_min_entropy(features_train,
                                          "EDUCATION",
                                          education_combinations)

print(entropy_education)
```

- Explanation

First I found the unique values for all the features, which are the categories that match up with the categories listed for each feature in the instructions of the homework. Next, I used the combinations library to find all the different combinations the categories of each feature can make up. Then I wrote a function that would calculate the entropy of when splitting the CAR_USE feature up by the different combinations I found earlier. Next, I wrote a function that would find the minimum entropy of all the entropy values I found earlier because the smaller the entropy, the more pure the values are within that split. I went on to put all the different combinations for each feature through the entropy function nadn the minimum entropy function so that I could find the optimal split for each feature.

Question 2c

- Answer

The feature selected for splitting in the first layer is OCCUPATION, where the branches are 'Blue Collar, Unknown, and Student' and all of the other occupations in the other branch. (Picked based on the lowest entropy value in part 2b)

The values are (4543, 2668), where there are 4543 values in the categories 'Blue Collar, Unknown, and Student' and 2668 values in all the other categories.

('Blue Collar', 'Unknown', 'Student')

- Entropy Value 0.71488
- values = (4543 2668)

- Code

```
#%% entropy for occupation feature

df_1_left = features_train[(features_train["OCCUPATION"] == "Blue Collar") |
                           (features_train["OCCUPATION"] == "Unknown") |
                           (features_train["OCCUPATION"] == "Student")]

df_1_right =  features_train[(features_train["OCCUPATION"] != "Blue Collar") &
                             (features_train["OCCUPATION"] != "Unknown") &
                             (features_train["OCCUPATION"] != "Student")]


print('Values in the branches of the first layer: (',
      len(df_1_right), ',', len(df_1_left), ')')
```

- Explanation

To find the feature selected for splitting in the first layer, I need to find the lowest entropy value in the options of optimal splits from (2b). The lowest minimum entropy of the optimal splits is the optimal split for OCCUPATION where entropy = 0.7148805225259208 and the values in the branches are 'Blue Collar', 'Unknown', and 'Student'. Next, to find the values in the branches of the first layer, I have to go through the features dataframe I established earlier with the features given and find which ones equal the categories of OCCUPATION that allow for the lowest entropy. So I went through the dataframe to ask for the number of values that are equal to ('Blue Collar', 'Unknown', 'Student') versus the ones that aren't equal to them for the right split value.

Question 2d

- Answer

Left branch/split:

- OCCUPATION: [('Blue Collar',), ('Unknown',), ('Student',)]
- EDUCATION: ('Below High School',)
    - entropy = 0.650587823999444
- CAR_TYPE: ('Minivan', 'SUV', 'Sports Car')
    - entropy = 0.7689481386570244

Right branch/split:

- ('Minivan', 'SUV', 'Sports Car')
    - entropy = 0.3212873372854656
- ('Below High School', 'High School', 'Bachelors')
    - entropy = 0.623532570928089
- ('Doctor', 'Lawyer')
    - entropy = 0.5740628685071641

- Code

```
#%% entropy calculations for left split if OCCUPATION picked

left_edu_entropy = calculate_min_entropy(df_1_left,
                                         "EDUCATION",
                                         education_combinations)

print(left_edu_entropy)

left_ct_entropy = calculate_min_entropy(df_1_left,
                                        "CAR_TYPE",
                                        car_type_combinations)

print(left_ct_entropy)
```

```
#%% entropy calculations for right split

occupation_column = ['Professional', 'Manager', 'Clerical', 'Doctor','Lawyer','Home Maker']
occupation_combinations = []
for i in range(1,math.ceil(len(occupation_column)/2)):
    occupation_combinations+=list(combinations(occupation_column,i))
right_occupation_entropy = calculate_min_entropy(df_1_right,"OCCUPATION",occupation_combinations)


right_edu_entropy = calculate_min_entropy(df_1_right,"EDUCATION",education_combinations)
right_ct_entropy = calculate_min_entropy(df_1_right,"CAR_TYPE",car_type_combinations)

print(right_ct_entropy , right_edu_entropy , right_occupation_entropy)
```

- Explanation

To get the features for splitting in the second layer, I used the dataframe from the left branch and calculated the entropy values for all the combinations of categories of features left over. Then I found the minimum entropy to know which feature to pick. I did the same process but with the dataframe from the right branch.

Question 2e

- Answer

Decision Rules	Counts [private, commercial]	Number of samples	Predicted probability [commercial, private]
(Occupation = (Blue Collar OR Student OR Unknown)) AND (Education = Below High School)	[140, 428]	568	[0.24647887323943662, 0.7535211267605634]
(Occupation = (Blue Collar OR Student OR Unknown)) AND (Education != Below High School)	[1786, 314]	2100	[0.8504761904761905, 1495238095238095]
(Occupation != (Blue Collar OR Student OR Unknown)) AND (Car type = (Minivan OR SUV OR Sports Car))	[20, 3231]	3251	[0.006151953245155337, 0.9938480467548446]
(Occupation != (Blue Collar OR Student OR Unknown)) AND (Car type != (Minivan OR SUV OR Sports Car))	[706, 586]	1292	[0.5464396284829721, 0.4535603715170279]

- Code

```
#%% second level of nodes started

cnt = 0
for i in df_2_left_left["Labels"]:
    if i == "Commercial":
        cnt+=1
proba_commercial = cnt/len(df_2_left_left["Labels"])
print("Count of commercial and private is",cnt,
      (len(df_2_left_left)-cnt),
      "respectively and probability of the event",
      proba_commercial)
```

```
#%%

cnt = 0
for i in df_2_left_right["Labels"]:
    if i == "Commercial":
        cnt+=1
proba_commercial = cnt/len(df_2_left_right["Labels"])
print("Count of commercial and private is",cnt,
      (len(df_2_left_right)-cnt),
      "respectively and probability of the event",
      proba_commercial)
```

```
cnt = 0
for i in df_2_right_left["Labels"]:
    if i == "Commercial":
        cnt+=1
proba_commercial = cnt/len(df_2_right_left["Labels"])
1-proba_commercial
print("Count of commercial and private is",cnt,
      (len(df_2_right_left)-cnt),
      "respectively and probability of the event",
      proba_commercial)
```

```
#%%

cnt = 0
for i in df_2_right_right["Labels"]:
    if i == "Commercial":
        cnt+=1
proba_commercial = cnt/len(df_2_right_right["Labels"])
proba_commercial
print("Count of commercial and private is",cnt,
      (len(df_2_right_right)-cnt),
      "respectively and probability of the event",
      proba_commercial)
```

- Explanation

To calculate the predicted probabilities for the leaf nodes of commercial vehicles, I divided the number of commercial vehicles by the number of vehicles in that dataframe.
To calculate the predicted probabilities for the leaf nodes of private vehicles, I divided the number of private vehicles by the number of vehicles in that dataframe.

Question 3a

- Answer

Below is the generated frequency table I got for the categorical target field.

```
3    4194
2    3532
1    2274
Name: y, dtype: int64
```

- Code

```
df = pandas.read_csv('/Users/tiffwong/Desktop/cs4

nObs = df.shape[0]

# Specify y as a categorical variable
y = df['y'].astype('category')
y_category = y.cat.categories
print(df['y'].value_counts())
```

- Explanation

I imported the sample_v10.csv file as the variable df and then using the library value_counts() to find the frequency of the values in the ‘y’ column.

Question 3b

- Answer

Initial model: y=Intercept+x1+x2+x3+x4+x5+x6+x7+x8+x9+x10
Model Log-Likelihood Value = -1956.0551397480979
Number of Free Parameters = 22

- Code

```
# Backward Selection
# Consider Model 0 is Origin = Intercept + all xi's
DriveTrain = df[['y']].astype('category')
X = pandas.get_dummies(DriveTrain)
X = df[all_x]
X = stats.add_constant(X, prepend=True)
DF1 = numpy.linalg.matrix_rank(X) * (len(y_category) - 1)

logit = stats.MNLogit(y, X)
thisFit = logit.fit(method='newton', full_output = True, maxiter = 100, tol = 1e-8)
thisParameter = thisFit.params
LLK1 = logit.loglike(thisParameter.values)

print(thisFit.summary())
print("Model Log-Likelihood Value =", LLK1)
print("Number of Free Parameters =", DF1)
```

- Figure/Command Output

```
In [243]: runcell(3, '/Users/tiffwong/Desktop/CS484/Week 3/untitled0.py')
Optimization terminated successfully.
         Current function value: 0.195606
         Iterations 10
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                10000
Model:                        MNLogit   Df Residuals:                     9978
Method:                           MLE   Df Model:                           20
Date:                Tue, 09 Mar 2021   Pseudo R-squ.:                  0.8170
Time:                        20:10:35   Log-Likelihood:                -1956.1
converged:                       True   LL-Null:                       -10688.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
       y=2       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.0165      0.087     11.636      0.000       0.845       1.188
x1            -1.1172      0.058    -19.343      0.000      -1.230      -1.004
x2            -0.0175      0.026     -0.669      0.503      -0.069       0.034
x3             0.0103      0.018      0.586      0.558      -0.024       0.045
x4            -1.5573      0.041    -38.103      0.000      -1.637      -1.477
x5             0.0030      0.010      0.287      0.774      -0.018       0.024
x6             0.0163      0.009      1.822      0.068      -0.001       0.034
x7         -1.268e-07      0.007   -1.7e-05      1.000      -0.015       0.015
x8            -0.0134      0.007     -2.028      0.043      -0.026      -0.000
x9             0.0076      0.006      1.315      0.189      -0.004       0.019
x10            0.0072      0.009      0.804      0.421      -0.010       0.025
------------------------------------------------------------------------------
       y=3       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.4041      0.106      3.817      0.000       0.197       0.612
x1            -1.1685      0.071    -16.354      0.000      -1.309      -1.028
x2             0.0002      0.033      0.005      0.996      -0.064       0.064
x3            -0.0009      0.022     -0.041      0.968      -0.045       0.043
x4            -0.0218      0.027     -0.794      0.427      -0.075       0.032
x5            -0.0088      0.013     -0.671      0.503      -0.034       0.017
x6             0.0004      0.011      0.038      0.970      -0.021       0.022
x7            -0.0017      0.010     -0.179      0.858      -0.021       0.017
x8            -0.0072      0.008     -0.867      0.386      -0.024       0.009
x9             0.0024      0.007      0.324      0.746      -0.012       0.017
x10            1.3464      0.038     35.838      0.000       1.273       1.420
==============================================================================
Model Log-Likelihood Value = -1956.0551397480979
Number of Free Parameters = 22
```

- Explanation

I ran the backward selection code with an array of all the variables: ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10'] using the model y = Intercept + all x_i's.

Question 3c

- Answer/Command Output

```
Removed feature: x1
Optimization terminated successfully.
         Current function value: 0.222527
         Iterations 10
Model Log-Likelihood Value = -2225.2713497027926
Number of Free Parameters = 20
Deviance = 538.4324199093894
Deviance degree of freedom = 2
Deviance significance value = 1.2047184950794192e-117

Removed feature: x2
Optimization terminated successfully.
         Current function value: 0.195633
         Iterations 10
Model Log-Likelihood Value = -1956.3310606384975
Number of Free Parameters = 20
Deviance = 0.5518417807993501
Deviance degree of freedom = 2
Deviance significance value = 0.7588729625234284

Removed feature: x3
Optimization terminated successfully.
         Current function value: 0.195628
         Iterations 10
Model Log-Likelihood Value = -1956.2803627435144
Number of Free Parameters = 20
Deviance = 0.45044599083303183
Deviance degree of freedom = 2
Deviance significance value = 0.7983381731549947

Removed feature: x4
Optimization terminated successfully.
         Current function value: 0.578049
         Iterations 9
Model Log-Likelihood Value = -5780.4943649849465
Number of Free Parameters = 20
Deviance = 7648.878450473698
Deviance degree of freedom = 2
Deviance significance value = 0.0

Removed feature: x5
Optimization terminated successfully.
         Current function value: 0.195647
         Iterations 10
Model Log-Likelihood Value = -1956.4701936937226
Number of Free Parameters = 20
Deviance = 0.8301078912495541
Deviance degree of freedom = 2
Deviance significance value = 0.6603046591967958
```

```
Removed feature: x6
Optimization terminated successfully.
         Current function value: 0.195808
         Iterations 10
Model Log-Likelihood Value = -1958.0835552791132
Number of Free Parameters = 20
Deviance = 4.0568310620305965
Deviance degree of freedom = 2
Deviance significance value = 0.1315437831571183

Removed feature: x7
Optimization terminated successfully.
         Current function value: 0.195607
         Iterations 10
Model Log-Likelihood Value = -1956.0744283318356
Number of Free Parameters = 20
Deviance = 0.03857716747552331
Deviance degree of freedom = 2
Deviance significance value = 0.9808962506876956

Removed feature: x8
Optimization terminated successfully.
         Current function value: 0.195811
         Iterations 10
Model Log-Likelihood Value = -1958.114764619672
Number of Free Parameters = 20
Deviance = 4.11924974314843
Deviance degree of freedom = 2
Deviance significance value = 0.12750179047077537

Removed feature: x9
Optimization terminated successfully.
         Current function value: 0.195695
         Iterations 10
Model Log-Likelihood Value = -1956.9532307012523
Number of Free Parameters = 20
Deviance = 1.7961819063089024
Deviance degree of freedom = 2
Deviance significance value = 0.4073465616020978

Removed feature: x10
Optimization terminated successfully.
         Current function value: 0.811114
         Iterations 7
Model Log-Likelihood Value = -8111.136792127912
Number of Free Parameters = 20
Deviance = 12310.163304759628
Deviance degree of freedom = 2
Deviance significance value = 0.0
```

- Code

```
for i in range(len(x_combos)):

    print("Removed feature:", all_x[i])
    y = df[['y']].astype('category')
    X = pandas.get_dummies(y)
    X = df[x_combos[i]]
    X = stats.add_constant(X, prepend=True)
    DF0 = numpy.linalg.matrix_rank(X) * (len(y_category) - 1)

    logit = stats.MNLogit(y, X)
    thisFit = logit.fit(method='newton', full_output = True, maxiter = 100, tol = 1e-8)
    thisParameter = thisFit.params
    LLK0 = logit.loglike(thisParameter.values)

    Deviance = 2 * (LLK1 - LLK0)
    DF = DF1 - DF0
    pValue = scipy.stats.chi2.sf(Deviance, DF)

    print("Model Log-Likelihood Value =", LLK0)
    print("Number of Free Parameters =", DF0)
    print("Deviance =", Deviance)
    print("Deviance degree of freedom =", DF)
    print("Deviance significance value =", pValue)
    print(' ')
```

- Explanation

I created a for loop that'll use all the combos on dataframes that remove one feature at a time, starting from x1 to x10. It calculates the name of the removed feature, the log-likelihood value of the reduced model, the number of free parameters of the reduced model, the Deviance test statistic, the Deviance degree of freedom, and the Deviance significance value. I used the formula of degree of freedom from the slides in class.

Question 3d

- Answer

The final model suggested by the Backward Selection method is y = Intercept + x1 + x4 + x10.

- Explanation

The rules of the Deviance Test Decision for Backwards Selection are:

1. Consider only predictors whose significances are greater than 5%
2. Remove the predictor which has the highest significance value from the current model

With these in mind, I can see that only when x1, x4, and x7 are removed are the ‘deviance significance value’ values less than 0.05, where x1 = 1.2047184950794192e-117, x4 = 0.0, and x7 = 0.03857716747552331.

Question 3e

- Answer

The Akaike Information Criterion and the Bayesian Information Criterion both suggest model #7, which is the model with the feature x7 removed.

- Code

```
  AIC.append(2.0 * DF0 - 2.0 * LLK0)
  BIC.append(DF0 * np.log(nObs) - 2.0 * LLK0)
```

```
for i in range(len(x_combos)):
    print('Removed feature:', all_x[i])
    print('Akaike Information Criterion:', AIC[i])
    print('Bayesian Information Criterion', BIC[i])
    print(' ')
```

```
print('AIC recommends: model #', AIC.index(min(AIC))+1)
print('BIC recommends: model #', BIC.index(min(BIC))+1)
```

- Figure/Command Output

```
Removed feature: x1
Akaike Information Criterion: 4490.542699405585
Bayesian Information Criterion 4634.749506845109

Removed feature: x2
Akaike Information Criterion: 3952.662121276995
Bayesian Information Criterion 4096.868928716519

Removed feature: x3
Akaike Information Criterion: 3952.560725487029
Bayesian Information Criterion 4096.767532926552

Removed feature: x4
Akaike Information Criterion: 11600.988729969893
Bayesian Information Criterion 11745.195537409416

Removed feature: x5
Akaike Information Criterion: 3952.9403873874453
Bayesian Information Criterion 4097.147194826969

Removed feature: x6
Akaike Information Criterion: 3956.1671105582263
Bayesian Information Criterion 4100.37391799775

Removed feature: x7
Akaike Information Criterion: 3952.1488566636713
Bayesian Information Criterion 4096.355664103195

Removed feature: x8
Akaike Information Criterion: 3956.229529239344
Bayesian Information Criterion 4100.4363366788675

Removed feature: x9
Akaike Information Criterion: 3953.9064614025046
Bayesian Information Criterion 4098.113268842028

Removed feature: x10
Akaike Information Criterion: 16262.273584255823
Bayesian Information Criterion 16406.480391695346
```

```
AIC recommends: model # 7
BIC recommends: model # 7
```

- Explanation

First I added two lines of code in my for loop previously that contains all the Akaike Information Criterion and the Bayesian Information Criterion in two separate arrays. Next, I printed all those values using a for loop so that I can have them all listed along with the models of which feature was removed for it. Then, I found the minimum of each AIC and BIC array so I could find the model that each criterion recommends. This is because in class, we learned that with these two models, I'm supposed to 'prefer a model that has a lower AIC value' and 'prefer a model that has a lower BIC value' (as stated in the slides). Finally I printed the index of the model in the two different arrays that are the minimum of the array so I know which model it refers to.