BuildADecisionTree

August 8, 2023

1 Assignment 3: Building a Decision Tree After Feature Transformations

[5]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

In this assignment, you will implement the following steps to build a Decision Tree classification model:

1. Load the "cell2celltrain" data set
2. Convert categorical features to one-hot encoded values
3. Split the data into training and test sets
4. Fit a Decision Tree classifier and evaluate the accuracy of its predictions
5. Plot the training set accuracy

1.1 Part 1. Load the Data Set

We will work with the "cell2celltrain" data set.

[6]:
```
# Do not remove or edit the line below:
filename = os.path.join(os.getcwd(), "data", "cell2celltrain.csv")
```

Task: Load the data and save it to DataFrame `df`.

[7]:
```
# YOUR CODE HERE
df=pd.read_csv(filename)
```

Task: Display the shape of `df` -- that is, the number of records (rows) and variables (columns)

[8]:
```
# YOUR CODE HERE
print(df.shape)
```

```
(51047, 58)
```

For the purpose of this assignment, we will remove the `Married` column due to missing values

[9]:
```
df.drop(columns = ['Married'], inplace=True)
```

1.2 Part 2. One-Hot Encode Categorical Values

To implement a decision tree model, we must first transform the string-valued categorical features into numerical boolean values using one-hot encoding.

1.2.1 a. Find the Columns Containing String Values

[10]:
```
df.dtypes
```

[10]:
```
CustomerID                int64
Churn                      bool
ServiceArea              object
ChildrenInHH               bool
HandsetRefurbished         bool
HandsetWebCapable          bool
TruckOwner                 bool
RVOwner                    bool
HomeownershipKnown         bool
BuysViaMailOrder           bool
RespondsToMailOffers       bool
OptOutMailings             bool
NonUSTravel                bool
OwnsComputer               bool
HasCreditCard              bool
NewCellphoneUser           bool
NotNewCellphoneUser        bool
OwnsMotorcycle             bool
MadeCallToRetentionTeam    bool
CreditRating             object
PrizmCode                object
Occupation               object
MonthlyRevenue          float64
MonthlyMinutes          float64
TotalRecurringCharge    float64
DirectorAssistedCalls   float64
OverageMinutes          float64
RoamingCalls            float64
PercChangeMinutes       float64
PercChangeRevenues      float64
DroppedCalls            float64
BlockedCalls            float64
UnansweredCalls         float64
CustomerCareCalls       float64
ThreewayCalls           float64
```

```
ReceivedCalls                float64
OutboundCalls                float64
InboundCalls                 float64
PeakCallsInOut               float64
OffPeakCallsInOut            float64
DroppedBlockedCalls          float64
CallForwardingCalls          float64
CallWaitingCalls             float64
MonthsInService              float64
UniqueSubs                   float64
ActiveSubs                   float64
Handsets                     float64
HandsetModels                float64
CurrentEquipmentDays         float64
AgeHH1                       float64
AgeHH2                       float64
RetentionCalls               float64
RetentionOffersAccepted      float64
ReferralsMadeBySubscriber    float64
IncomeGroup                  float64
AdjustmentsToCreditRating    float64
HandsetPrice                 float64
dtype: object
```

Task: Add all of the column names whos values are of type 'object' to a list named `to_encode`.

[11]:
```
# YOUR CODE HERE
to_encode=[]
for column in df.columns:
    if df[column].dtype=='object':
        to_encode.append(column)
```

Let's take a closer look at the candidates for one-hot encoding:

[12]:
```
df[to_encode].nunique()
```

[12]:
```
ServiceArea     747
CreditRating      7
PrizmCode         4
Occupation        8
dtype: int64
```

For all of the columns except for `ServiceArea`, it should be straightforward to replace a given column with a set of several new binary columns for each unique value. However, let's first deal with the special case of `ServiceArea`.

1.2.2 b. One Hot-Encoding 'ServiceArea': The Top 10 Values

Take a look at the number of unique values of the `ServiceArea` column. There are two many unique values in the `ServiceArea` column to attempt to create a new binary indicator column per value! One thing we could do is to see if some of the values in `ServiceArea` are occurring

frequently. We will then one-hot encode just those frequent values.

Task: Get the top 10 most frequent values in 'ServiceArea' and store them in list `top_10_SA`.

[13]:
```
# YOUR CODE HERE
top_10_SA=df['ServiceArea'].value_counts().head(10).index.tolist()
```

Task: Write a `for` loop that loops through every value in `top_10_SA` and creates one-hot encoded columns, titled 'ServiceArea + '_' + <service area value>'. For example, there will be a column named 'ServiceArea_NYCBRO917'. Use the NumPy `np.where()`function to accomplish this.

[14]:
```
# YOUR CODE HERE
for value in top_10_SA:
    column_name='ServiceArea'+value
    df[column_name]=np.where(df['ServiceArea']==value,1,0)
```

Task: 1. Drop the original, multi-valued `ServiceArea` column from the DataFrame `df`. 2. Remove 'ServiceArea' from the `to_encode` list.

[15]:
```
# YOUR CODE HERE
df.drop('ServiceArea',axis=1,inplace=True)
to_encode.remove('ServiceArea')
```

[16]:
```
df.head()
```

[16]:

	CustomerID	Churn	ChildrenInHH	HandsetRefurbished	HandsetWebCapable	TruckOwner	RVOwner	HomeownershipKnown	BuysViaMailOrder	RespondsToMailOffers	...	ServiceAreaNYCBRO917	ServiceAreaHOUHOU281	ServiceAreaDALDAL214	ServiceAreaNYCMAN917	ServiceAreaAPCFCH703
0	3000002	True	False	False	True	False	False	True	True	True	...	0	0	0	0	0
1	3000010	True	True	False	False	False	False	True	True	True	...	0	0	0	0	0
2	3000014	False	True	False	False	False	False	False	False	False	...	0	0	0	0	0
3	3000022	False	False	False	True	False	False	True	True	True	...	0	0	0	0	0
4	3000026	True	False	False	False	False	False	True	True	True	...	0	0	0	0	0

```
   ServiceAreaDALFTW817  ServiceAreaSANSAN210  ServiceAreaAPCSIL301  \
0                     0                     0                     0
1                     0                     0                     0
2                     0                     0                     0
3                     0                     0                     0
4                     0                     0                     0

  ServiceAreaSANAUS512 ServiceAreaSFROAK510
0                    0                    0
1                    0                    0
2                    0                    0
3                    0                    0
4                    0                    0

[5 rows x 66 columns]
```

1.2.3 c. One Hot-Encoding all Remaining Columns: All Unique Values per Column

All other columns in `to_encode` have reasonably small numbers of unique values, so we are going to simply one-hot encode every unique value of those columns.

Task: In the code cell below, iterate over column names and create new columns for all unique values. 1. Use a loop to loop over the column names in `to_encode` 2. In the loop: 1. Use the Pandas `pd.get_dummies()` function and save the result to variable `temp_df` 2. Use `df.join` to join `temp_df` with DataFrame `df`

[17]:
```
# YOUR CODE HERE
for column in to_encode:
    temp_df=pd.get_dummies(df[column],prefix=column)
    df=df.join(temp_df)
```

[18]:
```
df.head()
```

[18]:
```
   CustomerID  Churn  ChildrenInHH  HandsetRefurbished  HandsetWebCapable  \
0     3000002   True         False               False               True
1     3000010   True          True               False              False
2     3000014  False          True               False              False
3     3000022  False         False               False               True
4     3000026   True         False               False              False

   TruckOwner  RVOwner  HomeownershipKnown  BuysViaMailOrder  \
0       False    False                True              True
1       False    False                True              True
2       False    False               False             False
3       False    False                True              True
4       False    False                True              True

   RespondsToMailOffers  ...  PrizmCode_Suburban  PrizmCode_Town  \
```

```
0                  True  ...                   1                0
1                  True  ...                   1                0
2                 False  ...                   0                1
3                  True  ...                   0                0
4                  True  ...                   0                0

   Occupation_Clerical  Occupation_Crafts  Occupation_Homemaker  \
0                    0                  0                     0
1                    0                  0                     0
2                    0                  1                     0
3                    0                  0                     0
4                    0                  0                     0

   Occupation_Other  Occupation_Professional  Occupation_Retired  \
0                 0                        1                   0
1                 0                        1                   0
2                 0                        0                   0
3                 1                        0                   0
4                 0                        1                   0

  Occupation_Self Occupation_Student
0               0                  0
1               0                  0
2               0                  0
3               0                  0
4               0                  0

[5 rows x 85 columns]
```

Task: Remove all the original columns from DataFrame `df`

[19]:
```
# YOUR CODE HERE
df=df.drop(to_encode,axis=1)
```

[20]:
```
df.columns
```

[20]:
```
Index(['CustomerID', 'Churn', 'ChildrenInHH', 'HandsetRefurbished',
       'HandsetWebCapable', 'TruckOwner', 'RVOwner', 'HomeownershipKnown',
       'BuysViaMailOrder', 'RespondsToMailOffers', 'OptOutMailings',
       'NonUSTravel', 'OwnsComputer', 'HasCreditCard', 'NewCellphoneUser',
       'NotNewCellphoneUser', 'OwnsMotorcycle', 'MadeCallToRetentionTeam',
       'MonthlyRevenue', 'MonthlyMinutes', 'TotalRecurringCharge',
       'DirectorAssistedCalls', 'OverageMinutes', 'RoamingCalls',
       'PercChangeMinutes', 'PercChangeRevenues', 'DroppedCalls',
       'BlockedCalls', 'UnansweredCalls', 'CustomerCareCalls', 'ThreewayCalls',
       'ReceivedCalls', 'OutboundCalls', 'InboundCalls', 'PeakCallsInOut',
       'OffPeakCallsInOut', 'DroppedBlockedCalls', 'CallForwardingCalls',
       'CallWaitingCalls', 'MonthsInService', 'UniqueSubs', 'ActiveSubs',
       'Handsets', 'HandsetModels', 'CurrentEquipmentDays', 'AgeHH1', 'AgeHH2',
```

```
       'RetentionCalls', 'RetentionOffersAccepted',
       'ReferralsMadeBySubscriber', 'IncomeGroup', 'AdjustmentsToCreditRating',
       'HandsetPrice', 'ServiceAreaNYCBRO917', 'ServiceAreaHOUHOU281',
       'ServiceAreaDALDAL214', 'ServiceAreaNYCMAN917', 'ServiceAreaAPCFCH703',
       'ServiceAreaDALFTW817', 'ServiceAreaSANSAN210', 'ServiceAreaAPCSIL301',
       'ServiceAreaSANAUS512', 'ServiceAreaSFROAK510',
       'CreditRating_1-Highest', 'CreditRating_2-High', 'CreditRating_3-Good',
       'CreditRating_4-Medium', 'CreditRating_5-Low', 'CreditRating_6-VeryLow',
       'CreditRating_7-Lowest', 'PrizmCode_Other', 'PrizmCode_Rural',
       'PrizmCode_Suburban', 'PrizmCode_Town', 'Occupation_Clerical',
       'Occupation_Crafts', 'Occupation_Homemaker', 'Occupation_Other',
       'Occupation_Professional', 'Occupation_Retired', 'Occupation_Self',
       'Occupation_Student'],
      dtype='object')
```

Check that the data does not contain any missing values. The absense of missing values is necessary for training a Decision Tree model.

[21]:
```
# YOUR CODE HERE
missing_values=df.isnull().any().any()
```

1.3 Part 3: Create Labeled Examples from the Data Set

Task: Create labeled examples from DataFrame `df`. In the code cell below carry out the following steps:

- Get the `Churn` column from DataFrame `df` and assign it to the variable `y`. This will be our label.
- Get all other columns from DataFrame `df` and assign them to the variable `X`. These will be our features.

[22]:
```
# YOUR CODE HERE
y=df['Churn']
X=df.drop('Churn',axis=1)
```

1.4 Part 4: Create Training and Test Data Sets

Task: In the code cell below create training and test sets out of the labeled examples.

1. Use Scikit-learn's `train_test_split()` function to create the data sets.
2. Specify:
    - A test set that is 30 percent (.30) of the size of the data set.
    - A seed value of '123'.

[23]:
```
# YOUR CODE HERE
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.
↪3,random_state=123)
```

Check that the dimensions of the training and test datasets are what you expected:

[24]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(35732, 81)
(15315, 81)
```

1.5 Part 5. Fit a Decision Tree Classifer and Evaluate the Model

The code cell below contains a shell of a function named `train_test_DT()`. This function should train a Decision Tree classifier on the training data, test the resulting model on the test data, and compute and return the accuracy score of the resulting predicted class labels on the test data.

Task: Complete the function to make it work.

[25]:
```
def train_test_DT(X_train, X_test, y_train, y_test, leaf, depth,␣
↪crit='entropy'):
    '''
    Fit a Decision Tree classifier to the training data X_train, y_train.
    Return the accuracy of resulting predictions on the test set.
    Parameters:
        leaf := The minimum number of samples required to be at a leaf node
        depth := The maximum depth of the tree
        crit := The function to be used to measure the quality of a split.␣
↪Default: gini.
    '''

    # 1. Create the  Scikit-learn DecisionTreeClassifier model object below␣
↪and assign to variable 'model'
     # YOUR CODE HERE
  ␣
↪model=DecisionTreeClassifier(criterion=crit,max_depth=depth,min_samples_leaf=leaf)

    # 2. Fit the model to the training data below
     # YOUR CODE HERE
    model.fit(X_train,y_train)

    # 3. Make predictions on the test data and assign the result to the␣
↪variable 'class_label_predictions' below
     # YOUR CODE HERE
    class_label_predictions=model.predict(X_test)

    # 4. Compute the accuracy and save the result to the variable 'acc_score'␣
↪below
     # YOUR CODE HERE
    acc_score=accuracy_score(y_test,class_label_predictions)

    return acc_score
```

1.5.1 Train on Different Hyperparameter Values

Task: Train two Decision Tree classifiers using your function.

- one with a low value of depth
- one high value of depth

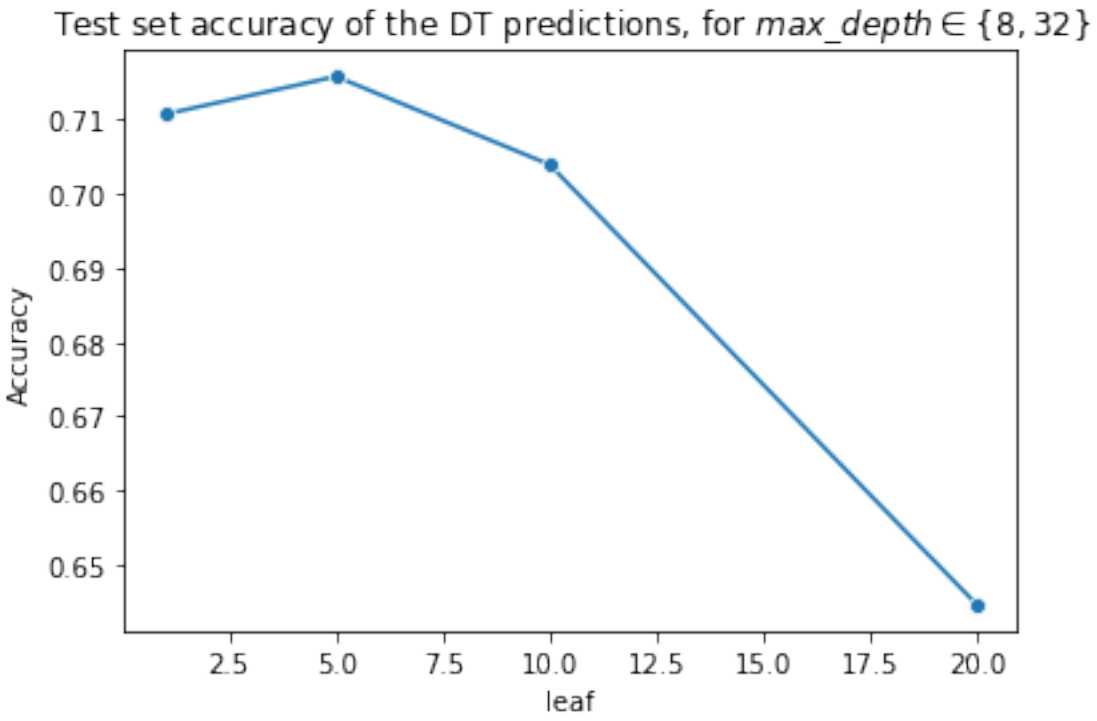
Specify the minimum number of samples at the leaf node to be equal to 1 for both trees. Save the resulting accuracy scores to list `acc`. Print the list.

[41]:
```
depth1= 3
depth2 = 10
leaf = 1

#max_depth_range = [3,5,8,10,12,15]#[depth1, depth2]
leaf_range=[1,5,10,20]
acc = []

# YOUR CODE HERE
for depth in leaf_range:
    accuracy=train_test_DT(X_train,X_test,y_train,y_test,leaf,depth)
    acc.append(accuracy)

print(acc)
```

```
[0.7107411034933072, 0.715834149526608, 0.7039503754489063, 0.6445968005223637]
```

Task: Visualize the results (Hint: use a `seaborn` lineplot).

[42]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
p = sns.lineplot(x=leaf_range,y=acc,marker='o')

plt.title('Test set accuracy of the DT predictions, for $max\_depth\in\{8,␣
 ↪32\}$')
ax.set_xlabel('leaf')
ax.set_ylabel('Accuracy')
plt.show()
```



Analysis: Experiment with different values for `max_depth`. Add these values to the list `max_depth_range` (i.e. change the values, create a list containing more values), retrain your model and rerun with the visualization cell above. Compare the different accuracy scores.

Once you find the best value for `max_depth`, experiment with different values for `leaf` and compare the different accuracy scores.

Is there one model configuration that yields the best score? Record your findings in the cell below.

The max depth with the highest accuracy is 5 I used a range from 3 to 15, with points at 3, 5, 8, 10, 12, and 15. Once I found this, I tested leaf values from 1 to 20 at points 1, 5, 10, and 20. Here, the highest accuracy score was also at 5. Not only were the points with the highest accuracy score the same, but so were the accuracy scores themselves as they were both at 0.715 = 71.5%.

[ ]:

[ ]: