Assignment Week 4

August 26, 2019

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
from collections import Counter
```

In [3]:
```
#Evaluation Metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
```

In [4]:
```
#Read the loans data
loans = pd.read_csv('lending-club-data/lending-club-data.csv')
```

```
/home/shrikrishna/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785:
  interactivity=interactivity, compiler=compiler, result=result)
```

In [5]:
```
train_index = pd.read_json('module-5-assignment-2-train-idx.json')
test_index = pd.read_json('module-5-assignment-2-test-idx.json')
```

In [6]:
```
train_data = loans.iloc[train_index[0], :]
test_data = loans.iloc[test_index[0], :]
```

In [7]:
```
pd.DataFrame(train_index).head()
```

Out[7]:
```
    0
0   1
1   6
2   7
3  10
4  12
```

In [8]:
```
train_data.shape
```

Out[8]: (37224, 68)

In [9]:
```
test_data.shape
```

Out[9]: (9284, 68)

0.0.1 Early stopping methods for decision trees

1. Reached a maximum depth. (set by parameter max_depth).
2. Reached a minimum node size. (set by parameter min_node_size).
3. Don't split if the gain in error reduction is too small. (set by parameter min_error_reduction).

0.1 Features

We will be considering only the following features

```
In [10]: [col for col in train_data.columns if(col.startswith('bad'))]

Out[10]: ['bad_loans']

In [11]: #Create a new column named 'safe_loans' using the column 'bad_loans'
         def create_safe(data):
             if('bad_loans' in data.columns):
                 data['safe_loans'] = data['bad_loans'].apply(lambda x:1 if(x==0) else 0)


In [12]: list(map(create_safe, [train_data, test_data]))

/home/shrikrishna/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  after removing the cwd from sys.path.


Out[12]: [None, None]

In [13]: features = ['grade',              # grade of the loan
                     'term',               # the term of the loan
                     'home_ownership',     # home_ownership status: own, mortgage or rent
                     'emp_length',         # number of years of employment
                    ]
         target = 'safe_loans'

In [14]: train_data[features + [target]].dtypes

Out[14]: grade              object
         term               object
         home_ownership     object
         emp_length         object
         safe_loans          int64
         dtype: object

In [15]: test_data[features + [target]].dtypes
```

```
Out[15]: grade                object
         term                 object
         home_ownership       object
         emp_length           object
         safe_loans            int64
         dtype: object
```

0.2 Missing Value Analysis

```
In [16]: def find_missing_values(data):
             features = data.columns

             missing_value_count = data.isna().sum()

             missing_value_percentage = data.isna().sum()*100/data.shape[0]

             missing_data = pd.DataFrame({'Features': features,
                                     'Missing Value Count': missing_value_count,
                                     'Missing Value Percentage': missing_value_percentage
                                     columns = ['Features', 'Missing Value Count',
                                                'Missing Value Percentage'])
             missing_data = missing_data.sort_values(by='Missing Value Percentage',
                                                ascending = False)

             return missing_data

In [17]: find_missing_values(train_data[features]).head()
```

Out[17]:

	Features	Missing Value Count	Missing Value Percentage
emp_length	emp_length	1443	3.876531
grade	grade	0	0.000000
term	term	0	0.000000
home_ownership	home_ownership	0	0.000000

```
In [18]: find_missing_values(test_data[features]).head()
```

Out[18]:

	Features	Missing Value Count	Missing Value Percentage
emp_length	emp_length	349	3.759156
grade	grade	0	0.000000
term	term	0	0.000000
home_ownership	home_ownership	0	0.000000

```
In [19]: set(train_data['emp_length'])

Out[19]: {'1 year',
          '10+ years',
          '2 years',
          '3 years',
          '4 years',
```

```
 '5 years',
 '6 years',
 '7 years',
 '8 years',
 '9 years',
 '< 1 year',
 nan}
```

In [20]:
```
#Replace the missing values in the 'emp_length' column with 0
train_data.loc[train_data['emp_length'].isna()==True, 'emp_length']  = '0'
```

```
/home/shrikrishna/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py:543: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  self.obj[item] = s
```

In [21]:
```
test_data.loc[test_data['emp_length'].isna()==True, 'emp_length'] = '0'
```

```
/home/shrikrishna/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py:543: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  self.obj[item] = s
```

In [22]:
```
find_missing_values(train_data[features]).head()
```

Out[22]:

	Features	Missing Value Count	Missing Value Percentage
grade	grade	0	0.0
term	term	0	0.0
home_ownership	home_ownership	0	0.0
emp_length	emp_length	0	0.0

In [23]:
```
find_missing_values(test_data[features]).head()
```

Out[23]:

	Features	Missing Value Count	Missing Value Percentage
grade	grade	0	0.0
term	term	0	0.0
home_ownership	home_ownership	0	0.0
emp_length	emp_length	0	0.0

In [24]:
```
set(train_data['emp_length'])
```

Out[24]:
```
{'0',
 '1 year',
```

```
           '10+ years',
           '2 years',
           '3 years',
           '4 years',
           '5 years',
           '6 years',
           '7 years',
           '8 years',
           '9 years',
           '< 1 year'}
```

```
In [25]: set(test_data['emp_length'])
```

```
Out[25]: {'0',
          '1 year',
          '10+ years',
          '2 years',
          '3 years',
          '4 years',
          '5 years',
          '6 years',
          '7 years',
          '8 years',
          '9 years',
          '< 1 year'}
```

0.3 Transform categorical data into binary features

```
In [26]: train_data[features].head()
```

Out[26]:

	grade	term	home_ownership	emp_length
1	C	60 months	RENT	< 1 year
6	F	60 months	OWN	4 years
7	B	60 months	RENT	< 1 year
10	C	36 months	RENT	< 1 year
12	B	36 months	RENT	3 years

```
In [27]: test_data[features].head()
```

Out[27]:

	grade	term	home_ownership	emp_length
24	D	60 months	RENT	2 years
41	A	36 months	MORTGAGE	10+ years
60	F	60 months	RENT	4 years
93	D	60 months	RENT	10+ years
132	B	36 months	RENT	2 years

```
In [28]: OHE_features = []
         def to_categorical(data, features = features):
             global OHE_features
```

```
    #Convert the categorical features to One-Hot-Encoded features
    df = pd.get_dummies(data[features])
    OHE_features = df.columns.tolist()

    #Append the new encoded data frame to the orignal data frame
    data = pd.concat([data, df], axis=1)

    #Drop the orignal categorical variables
    if(set(features).intersection() == set(features)):
        data = data.drop(features, axis=1)

    return (data)
```

In [29]:
```
#train_data.columns.tolist()
```

In [30]:
```
train_data, test_data = list(map(to_categorical, [train_data, test_data]))
```

In [31]:
```
#Check whether the old features are present in the data
set(train_data.columns).intersection(features) == set() and \
set(test_data.columns).intersection(features) == set()
```

Out[31]:
```
True
```

In [32]:
```
#Check whether the old features are present in the data
set(train_data.columns).intersection(features) != set() or \
set(test_data.columns).intersection(features) != set()
```

Out[32]:
```
False
```

In [33]:
```
train_data.head()
```

Out[33]:
```
         id  member_id  loan_amnt  funded_amnt  funded_amnt_inv  int_rate  \
1   1077430    1314167       2500         2500             2500     15.27
6   1071795    1306957       5600         5600             5600     21.28
7   1071570    1306721       5375         5375             5350     12.69
10  1064687    1298717       9000         9000             9000     13.49
12  1069057    1303503      10000        10000            10000     10.65

    installment sub_grade                         emp_title  annual_inc  \
1         59.83        C4                             Ryder     30000.0
6        152.39        F2                               NaN     40000.0
7        121.45        B5                         Starbucks     15000.0
10       305.38        C1  Va. Dept of Conservation/Recreation     30000.0
12       325.74        B2                             SFMTA    100000.0

             ...         emp_length_10+ years emp_length_2 years  \
1            ...                            0                  0
6            ...                            0                  0
7            ...                            0                  0
```

```
10          ...                                  0                  0
12          ...                                  0                  0

   emp_length_3 years emp_length_4 years emp_length_5 years  \
1                   0                  0                  0
6                   0                  1                  0
7                   0                  0                  0
10                  0                  0                  0
12                  1                  0                  0

   emp_length_6 years emp_length_7 years emp_length_8 years  \
1                   0                  0                  0
6                   0                  0                  0
7                   0                  0                  0
10                  0                  0                  0
12                  0                  0                  0

   emp_length_9 years emp_length_< 1 year
1                   0                   1
6                   0                   0
7                   0                   1
10                  0                   1
12                  0                   0

[5 rows x 90 columns]
```

In [34]:
```
target in test_data.columns
```

Out[34]:
```
True
```

In [35]:
```
binary_features = train_data.columns
```

In [36]:
```
target in train_data.columns
```

Out[36]:
```
True
```

In [37]:
```
np.unique(train_data[target])
```

Out[37]:
```
array([0, 1])
```

0.4 Calculate the Mistakes

In [38]:
```
def intermediate_node_num_mistakes(labels_in_node):
    if len(labels_in_node)==0:
        return (0)

    #Find the number of -1s
    neg_count = len([ele for ele in labels_in_node if(ele==-1)])
```

```
        #Find the number of 1s
        pos_count = len([ele for ele in labels_in_node if(ele==1)])

        # Return the number of mistakes that the majority classifier makes.

        if(neg_count>pos_count):
            max_class = -1
        else:
            max_class = 1

        #Get the prediction array
        predicted = np.array(len(labels_in_node) * [max_class])

        mistakes = 0
        for act, pred in zip(labels_in_node, predicted):
            if(act!=pred):
                mistakes += 1

        return mistakes
```

In [39]:
```
# Test case 1
example_labels = np.array([-1, -1, 1, 1, 1])
if intermediate_node_num_mistakes(example_labels) == 2:
    print ('Test passed!')
else:
    print ('Test 1 failed... try again!')

# Test case 2
example_labels = np.array([-1, -1, 1, 1, 1, 1, 1])
if intermediate_node_num_mistakes(example_labels) == 2:
    print ('Test passed!')
else:
    print ('Test 3 failed... try again!')

# Test case 3
example_labels = np.array([-1, -1, -1, -1, -1, 1, 1])
if intermediate_node_num_mistakes(example_labels) == 2:
    print ('Test passed!')
else:
    print ('Test 3 failed... try again!')
```

```
Test passed!
Test passed!
Test passed!
```

0.5 9. Follow these steps to implement best_splitting_feature:

```
Step 1: Loop over each feature in the feature list
```

Step 2: Within the loop, split the data into two groups: one group where all of the data has fe

Step 3: Calculate the number of misclassified examples in both groups of data and use the above

Step 4: If the computed error is smaller than the best error found so far, store this feature a

In [40]:
```
def split_on(feature):
    intermediate_node_num_mistakes(labels_in_node)
```

In [41]:
```
def best_splitting_feature(data, features, target):

    target_values = data[target]
    best_feature = None # Keep track of the best feature
    best_error = 10     # Keep track of the best error so far
    # Note: Since error is always <= 1, we should intialize it with something larger

    # Convert to float to make sure error gets computed correctly.
    num_data_points = float(len(data))

    # Loop through each feature to consider splitting on that feature
    for feature in features:

        # The left split will have all data points where the feature value is 0
        left_split = data.loc[data[feature] == 0, feature]

        # The right split will have all data points where the feature value is 1
        ## YOUR CODE HERE
        right_split = data.loc[data[feature] == 1, feature]

        # Calculate the number of misclassified examples in the left split.
        # Remember that we implemented a function for this! (It was called intermedia
        # YOUR CODE HERE
        left_mistakes = intermediate_node_num_mistakes(left_split)

        # Calculate the number of misclassified examples in the right split.
        ## YOUR CODE HERE
        right_mistakes = intermediate_node_num_mistakes(right_split)

        # Compute the classification error of this split.
        # Error = (# of mistakes (left) + # of mistakes (right)) / (# of data points)
        ## YOUR CODE HERE
        error = (left_mistakes + right_mistakes)/num_data_points

        # If this is the best error we have found so far, store the feature as best_f
        ## YOUR CODE HERE
        if error < best_error:
            best_error = error
```