Decision Tree Problem_final

February 15, 2019

1 CS 536 : Decision Trees Assignment

by Sagar Jain (sj735)

```
In [4]: #Importing libraries
        import numpy as np
        import pandas as pd
        from pprint import pprint
        import matplotlib.pyplot as plt
        plt.rcParams['figure.figsize'] = [15, 6]
```

```
In [29]: #Initializing global variables
         k = 4
         m = 8
         epsilon = 0.0000001
```

2 Answer 1

For a given value of k;m, (number of features, number of data points), write a function to generate a training data set based on the above scheme.

Function to generate rows >>

```
In [45]: def vector_generation_function(k=0):
                X = []
                Y = 0
                w_denom = sum([0.9**x for x in range(2,k+1)])

                for x in range(1, k+1):
                    if x != 1:
                        prev_X = X[-1]
                        X.append(int(np.random.choice([prev_X, 1-prev_X], 1, p=[0.75, 0.25])))
                    else: # when x == 1
                        X.append(int(np.random.choice([1,0], 1, p=[0.5, 0.5])))

                if sum([ ((0.9**n)/w_denom)*X[n-1] for n in range(2,k+1) ]) >= 1/2:
                    X.append(X[1])
                else:
```

```
            X.append(1 - X[1])

        return X
```

Function to generate dataset >>

```
In [67]: def dataframe_function(k=0, m=0):

             data = []

             for x in range(1, m+1):
                 data.append(vector_generation_function(k))

             #Create header list
             headers = ['X'+str(x) for x in range(1,k+1)] + ['Y']

             dataframe = pd.DataFrame(data, columns=headers)

             return dataframe
```

```
In [54]: #Generating data set
         dataframe = dataframe_function(4, 8)
         print(dataframe)
```

```
   X1  X2  X3  X4  Y
0   1   0   1   1  0
1   0   0   0   0  1
2   1   1   1   1  1
3   1   0   0   0  1
4   1   0   0   0  1
5   0   1   1   1  1
6   1   1   1   0  1
7   0   0   0   0  1
```

Function to calculate Information Gain >>

```
In [70]:  def information_gain(subset_dataframe):

              classes = list(subset_dataframe)
              x0_y0 = len(subset_dataframe.loc[(subset_dataframe[classes[0]] == 0) & (subse

              x0_y1 = len(subset_dataframe.loc[(subset_dataframe[classes[0]] == 0) & (subse

              x1_y0 = len(subset_dataframe.loc[(subset_dataframe[classes[0]] == 1) & (subse

              x1_y1 = len(subset_dataframe.loc[(subset_dataframe[classes[0]] == 1) & (subse

              py0 = (x0_y0 + x1_y0) / (x0_y0 + x1_y0 + x0_y1 + x1_y1 + epsilon)
```

```
        py1 = (x0_y1 + x1_y1) / (x0_y0 + x1_y0 + x0_y1 + x1_y1 + epsilon)

        px0 = (x0_y0 + x0_y1) / (x0_y0 + x1_y0 + x0_y1 + x1_y1 + epsilon)

        px1 = (x1_y0 + x1_y1) / (x0_y0 + x1_y0 + x0_y1 + x1_y1 + epsilon)

        py0_x0 = x0_y0 / (x0_y0 + x0_y1 + epsilon)

        py1_x0 = x0_y1 / (x0_y0 + x0_y1 + epsilon)

        py0x1 = x1_y0 / (x1_y0 + x1_y1 + epsilon)

        py1x1 = x1_y1 / (x1_y0 + x1_y1 + epsilon)

        pyx0 = (-1 * py0_x0 * np.log(py0_x0 + epsilon)) + (-1 * py1_x0 * np.log(py1_x0

        pyx1 = (-1 * py0x1 * np.log(py0x1 + epsilon)) + (-1 * py1x1 * np.log(py1x1 + e

        hy =  (-1 * py0 * np.log(py0 + epsilon)) + (-1 * py1 * np.log(py1 + epsilon))
        hyx = (px1 * pyx1) + (px0 * pyx0)

        igx = hy - hyx

        return igx
```

Function to split feature based on Information Gain >>

In [33]:
```
#Splitting Variable function
def splitting_variable(dataframe):
    """This function take a dataframe as input and returns apt splitting variable bas

    #Fetch information gain for every X
    columns = list(dataframe)
    ig = [information_gain(dataframe[[x, 'Y']]) for x in columns[:-1]]
    split_on_variable = columns[np.argmax(ig)]

    return split_on_variable #splitting variable
```

In [49]:
```
split_var = splitting_variable(dataframe)
split_var
```

Out[49]: 'X1'

3 Answer 2

Given a data set, write a function to
t a decision tree to that data based on splitting the variables by maximizing the information gain. Additionally, return the training error of this tree on the data set, err_train(f).

In [34]:
```
#generate decision tree
def generate_decision_tree(dataframe, tree=None):

    #fetch the features X1, X2...Xn
    classes = list(dataframe)[:-1]

    #get the node to split on
    split_var = splitting_variable(dataframe)

    #initialize tree in form of dictionary if not already initialized
    if tree is None:
        tree = {}
        tree[split_var] = {}

    #Explore when split_var is 0 & 1
    for value in (0,1):
        split_dataframe = dataframe[dataframe[split_var] == value]
        class_value, value_count = np.unique(split_dataframe['Y'], return_counts=True

        #check if split_dataframe has only single class to consider, if not then expl
        if len(value_count) == 1:
            tree[split_var][value] = class_value[0]
        else:
            #recursively call the tree
            tree[split_var][value] = generate_decision_tree(split_dataframe)

    #return the generated tree
    return tree
```

In [50]:
```
df = dataframe_function(k, 8)
tree = generate_decision_tree(df)
print(df)
pprint(tree)
```

```
   X1  X2  X3  X4  X5  X6  X7  X8  X9  X10  Y
0   1   1   1   0   0   0   1   1   1    1  1
1   0   0   0   1   1   1   0   0   1    1  0
2   0   0   0   0   0   0   0   0   1    0  1
3   0   0   1   1   1   0   0   0   0    0  1
4   0   0   0   0   0   0   0   1   1    1  1
5   1   0   1   1   0   1   1   1   0    1  0
6   0   0   0   0   1   1   1   1   1    1  0
7   0   0   0   0   1   1   0   0   0    0  1
{'X6': {0: 1, 1: {'X10': {0: 1, 1: 0}}}}
```

In [51]:
```
#predict function using the decision tree
def predict(X_dataframe, tree):
```

```
            for value in tree.keys():
                sub_tree = int(X_dataframe[value])
                tree = tree[value][sub_tree]

                if type(tree) is not dict:
                    return tree
                else:
                    return predict(X_dataframe, tree)
```

```
In [52]: X_df = dataframe_function(k, 1)
         print(X_df)
         predict(X_df, tree)
```

```
   X1  X2  X3  X4  X5  X6  X7  X8  X9  X10  Y
0   1   1   1   1   1   1   1   0   0    1  1
```

```
Out[52]: 0
```

```
In [36]: #Create function for computing ERR_train
         def compute_ERR(dataframe, tree):

             err = 0
             length = len(dataframe)

             for row_index in range(0, length):
                 row = dataframe[row_index: row_index+1]

                 if int(row['Y']) != predict(row, tree):
                     err += 1

             return err/length
```

4 Answer 3

For k = 4 and m = 30, generate data and
t a decision tree to it. Does the ordering of the variables in the decision tree make sense, based on the function that de
nes Y ? Why or why not? Draw the tree.

```
In [56]: k = 4
         m = 30

         #Generate dataframe based on above parameters
         df = dataframe_function(k, m)
         print('Data Set:\n')
         print(df,'\n')
```

```python
#Fit decision tree
tree = generate_decision_tree(df)
print('\nDecision Tree: \n')
pprint(tree)
print('\n')

#Compute Training Error
ERR_train = compute_ERR(df, tree)
print('Train Error:')
print(ERR_train,'\n')
```

```
Data Set:

    X1  X2  X3  X4  Y
0    1   1   1   1  1
1    0   0   0   0  1
2    0   0   1   1  0
3    0   0   1   0  1
4    1   0   0   1  1
5    0   1   0   0  0
6    0   0   0   1  1
7    0   1   1   1  1
8    0   0   0   0  1
9    0   0   0   0  1
10   0   0   0   0  1
11   0   0   0   0  1
12   0   1   1   1  1
13   0   0   0   1  1
14   0   0   0   0  1
15   1   1   1   1  1
16   1   0   0   0  1
17   0   1   0   0  0
18   1   1   1   1  1
19   1   1   1   0  1
20   0   0   1   0  1
21   1   1   1   0  1
22   0   0   0   0  1
23   0   0   0   1  1
24   0   0   1   1  0
25   0   1   1   1  1
26   0   0   1   1  0
27   0   0   0   0  1
28   0   1   0   0  0
29   0   0   0   0  1


Decision Tree:
```

```
{'X1': {0: {'X2': {0: {'X3': {0: 1, 1: {'X4': {0: 1, 1: 0}}}},
                   1: {'X3': {0: 0, 1: 1}}}},
        1: 1}}
```

```
Train Error:
0.0
```

Yes definitely, the order of the variables in the decision tree makes sense. We can justify this by checking the training error of generated decision tree which is 0.0 that means that the decision tree models our training data perfectly.

5 Answer 4

Write a function that takes a decision tree and estimates its typical error on this data err(f); i.e., generate a lot of data according to the above scheme, and

nd the average error rate of this tree over that data.

In [38]:
```
k = 4
m = 10

#Generate dataframe based on above parameters
df = dataframe_function(k, m)
print('Data Set:\n')
print(df,'\n')

#Compute typical error on above created data and tree drawn above
ERR = compute_ERR(df, tree)
print('Average Error:')
print(ERR,'\n')
```

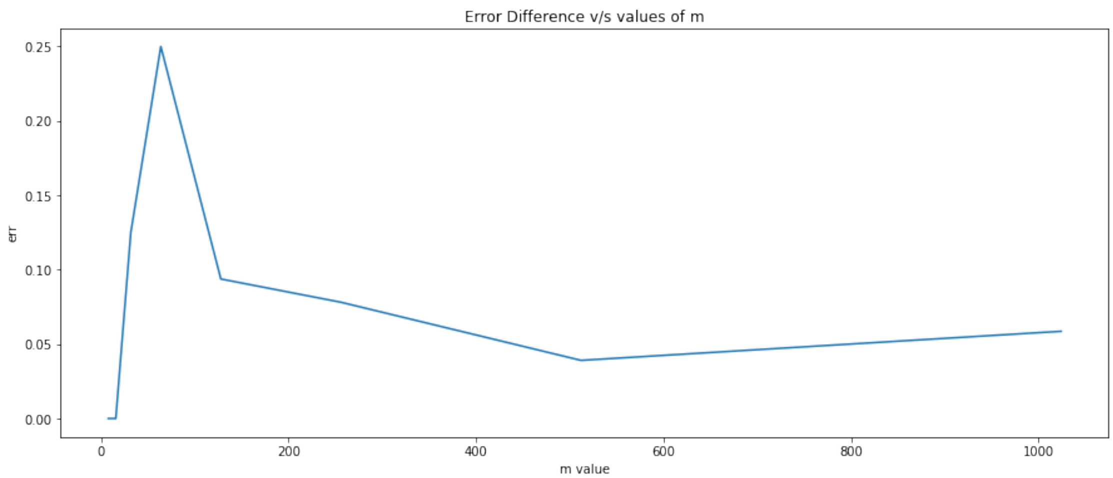
```
Data Set:

   X1  X2  X3  X4  Y
0   1   1   0   0  0
1   1   1   0   0  0
2   1   1   1   1  1
3   1   0   0   1  1
4   1   0   0   1  1
5   0   0   1   0  1
6   0   1   0   1  1
7   0   0   0   0  1
8   1   1   1   1  1
9   0   1   1   1  1

Average Error:
```

```
0.3
```

6 Answer 5

For k = 10, estimate the value of |err_train(f) - err(f)| for a given m by repeatedly generating data sets, fitting trees to those data sets, and estimating the true and training error. Do this for multiple m, and graph this difference as a function of m. What can you say about the marginal value of additional training data?

In [79]:
```
k = 10
m = [8, 16, 32, 64, 128, 256, 512,1024]
split_ratio = 0.75
err_diff = {}

for values in m:
    df = dataframe_function(k, values)
    train_df = df.iloc[:int(split_ratio*values)]
    test_df = df.iloc[int(split_ratio*values):]

    tree = generate_decision_tree(train_df)

    ERR_train = compute_ERR(train_df, tree)
    ERR = compute_ERR(test_df, tree)

    err_diff[values] = np.absolute(ERR_train - ERR)

err_list = sorted(err_diff.items())
x,y = zip(*err_list)

plt.plot(x,y)
plt.title('Error Difference v/s values of m')
plt.xlabel('m value')
plt.ylabel('err')
plt.show()
```



the marginal value of additional training data? We can observe that on increasing the value of m, the error difference minimises contineously.

7 Answer 6

Design an alternative metric for splitting the data, not based on information content / information gain. Repeat the computation from (5) above for your metric, and compare the performance of your trees vs the ID3 trees.

Function to split feature based on Chi-Squared test >>

Lets use Chi-Squared test to check the weight of dependence of feature variables with Y and split based on the most dependent feature first.

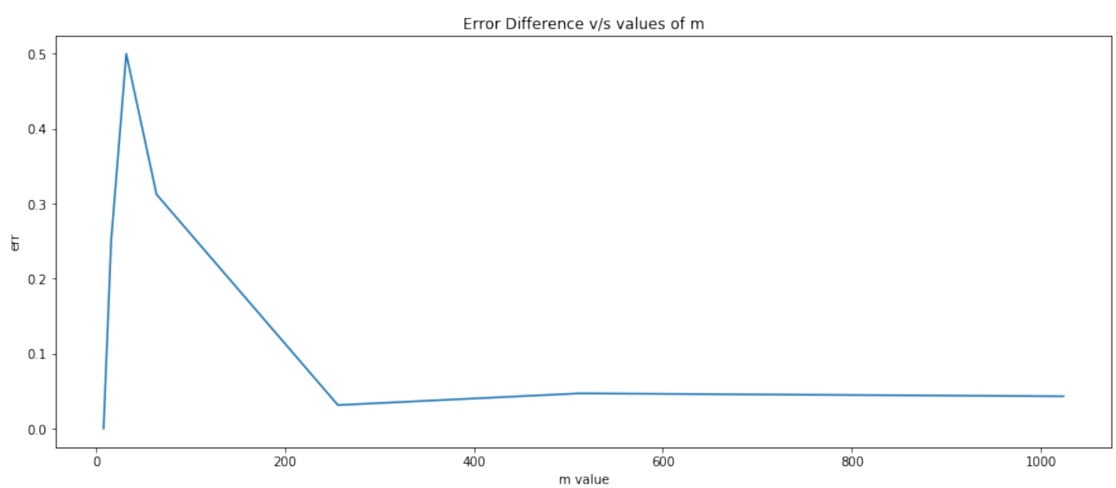
```
In [ ]: #Splitting function based on Chi-Squared Test
        def chi_squared_function(subset_df):

                classes = list(subset_df)
                x0y0 = len(subset_df.loc[(subset_df[classes[0]] == 0) & (subset_df[classes[1]]

                x0y1 = len(subset_df.loc[(subset_df[classes[0]] == 0) & (subset_df[classes[1]]

                x1y0 = len(subset_df.loc[(subset_df[classes[0]] == 1) & (subset_df[classes[1]]

                x1y1 = len(subset_df.loc[(subset_df[classes[0]] == 1) & (subset_df[classes[1]]

                total_count = x0y0 + x0y1 + x1y0 + x1y1

                py0 = (x0y0 + x1y0) / (total_count + epsilon)
                py1 = (x0y1 + x1y1) / (total_count + epsilon)

                px0 = (x0y0 + x0y1) / (total_count + epsilon)
                px1 = (x1y0 + x1y1) / (total_count + epsilon)
```

```
            t_x0y0 = (((px0 * py0 * total_count) - x0y0) ** 2)/((px0 * py0 * total_count) 

            t_x0y1 = (((px0 * py1 * total_count) - x0y1) ** 2)/((px0 * py1 * total_count) 

            t_x1y0 = (((px1 * py0 * total_count) - x1y0) ** 2)/((px1 * py0 * total_count) 

            t_x1y1 = (((px1 * py1 * total_count) - x1y1) ** 2)/((px1 * py1 * total_count) 

            T = t_x0y0 + t_x0y1 + t_x1y0 + t_x1y1

            return T


        def chi_squared_split(dataframe):

            #Fetch information gain for every X
            columns = list(dataframe)
            chi = [chi_squared_function(dataframe[[x, 'Y']]) for x in columns[:-1]]

            split_on_variable = columns[np.argmax(chi)]

            return split_on_variable #splitting variable
```

In [72]:
```
#generate decision tree using chi-squared
def decision_tree_chi_generation(dataframe, tree=None):

    #fetch the features X1, X2...Xn
    classes = list(dataframe)[:-1]

    #get the node to split on
    split_var = chi_squared_split(dataframe)

    #initialize tree in form of dictionary if not already initialized
    if tree is None:
        tree = {}
        tree[split_var] = {}

    #Explore when split_var is 0 & 1
    for value in (0,1):
        split_dataframe = dataframe[dataframe[split_var] == value]
        class_value, value_count = np.unique(split_dataframe['Y'], return_counts=True

        #check if split_dataframe has only single class to consider, if not then expl
        if len(value_count) == 1:
            tree[split_var][value] = class_value[0]
        else:
            #recursively call the tree
```

```
                tree[split_var][value] = decision_tree_chi_generation(split_dataframe)

        #return the generated tree
        return tree
```

In [80]:
```
#Question 6
k = 10
m = [8, 16, 32, 64, 128, 256, 512,1024]
split_ratio = 0.75
err_diff = {}

for values in m:
    df = dataframe_function(k, values)
    train_df = df.iloc[:int(split_ratio*values)]
    test_df = df.iloc[int(split_ratio*values):]

    tree = decision_tree_chi_generation(train_df)

    ERR_train = compute_ERR(train_df, tree)
    ERR = compute_ERR(test_df, tree)

    err_diff[values] = np.absolute(ERR_train - ERR)

err_list = sorted(err_diff.items())
x,y = zip(*err_list)

plt.plot(x,y)
plt.title('Error Difference v/s values of m')
plt.xlabel('m value')
plt.ylabel('err')
plt.show()
```



8 Comparing the performance of generated trees vs the ID3 trees.

On the basis of graph, we can make an observation that chi square performed better than the trivial decision tree method for high value of m.