# CS 536 : Pruning Decision Trees 16:198:536

**Suyu Huang**
**189006098**

***1) Write a function to generate m samples of (X, Y ), and another to fit a tree to that data using ID3. Write a third function to, given a decision tree f, estimate the error rate of that decision tree on the underlying data, err(f). Do this repeatedly for a range of m values, and plot the 'typical' error of a tree trained on m data points as a function of m. Does this agree with your intuition?***

**Solution:**

The code of the function to generate m sample:

```
def generateDataSet(m):
    k = 21
    data = [[0] * (k + 1) for i in range(m)]
    labels = ['X' + str(i) for i in range(k)]

    for i in range(m):
        for j in range(k):
                    if j == 0 :
                        data[i][j] = 0 if random.uniform(0, 1) < 0.5 else 1
                    elif j > 0 and j <= 14:
                        data[i][j] = data[i][j - 1] if random.uniform(0, 1) < 0.75 else (1 - data[i][j - 1])
                    else:
                        data[i][j] = 0 if random.uniform(0, 1) < 0.5 else 1
    for rowVector in data:
        if rowVector[0] == 0:
            X_1_to_7 = rowVector[1:8]
            rowVector[-1] = 0 if X_1_to_7.count(0) > 3 else 1
        else:
            X_8_to_14 = rowVector[8:15]
            rowVector[-1] = 1 if X_8_to_14.count(1) > 3 else 0
    return data, labels
```

The function to fit a tree to that data using ID3:

```
def buildDecisionTree(dataSet, labelCopy):
    Y = [rowVector[-1] for rowVector in dataSet]
    label = labelCopy[:]
    # return the leave node
    # if Y can only be 1 or 0, return the value, no need to split data
    if Y.count(Y[0]) == len(Y):
        return Y[0]

    if len(dataSet[0]) == 1:
        countYEqualTo1 = sum(Y)
        ans = 1 if countYEqualTo1 >= (len(Y) / 2) else 0
        return ans

    maxIndexOfInfoGain = maximumInformationGainIndex(dataSet)
    maxIndex_Label = label[maxIndexOfInfoGain]
    dcTree = {maxIndex_Label: {}}
    del(label[maxIndexOfInfoGain])

    selectedCol = [rowVector[maxIndexOfInfoGain] for rowVector in dataSet]
    setOfselectedCol = set(selectedCol)
    for val in setOfselectedCol:
        copyOflable = label[:]
        subDataSet = splitDataSet(dataSet, maxIndexOfInfoGain, val)
        dcTree[maxIndex_Label][val] = buildDecisionTree(subDataSet, copyOflable)
```

The third function that given a tree, estimate the error train:

```
def calculateError(tree, mrows):
    tError = 0
    newDataSet, labels = generateDataSet(mrows)
    for rowVector in newDataSet:
        tError += isError(tree, rowVector)
    return float(tError) / mrows

def isError(tree, rowVector):
    k = len(rowVector) - 1
    lables = ['X' + str(i) for i in range(k)]

    while tree != 1 and tree != 0:
        label = list(tree.keys())
        index = lables.index(label[0])
        value = rowVector[index]
        if value in tree[label[0]]:
            tree = tree[label[0]][value]
        else: return 1
    return 0 if tree == rowVector[-1] else 1
```

Do this repeatedly for a range of m values, and plot the ‘typical’ error of a tree trained on m data points as a function of m:

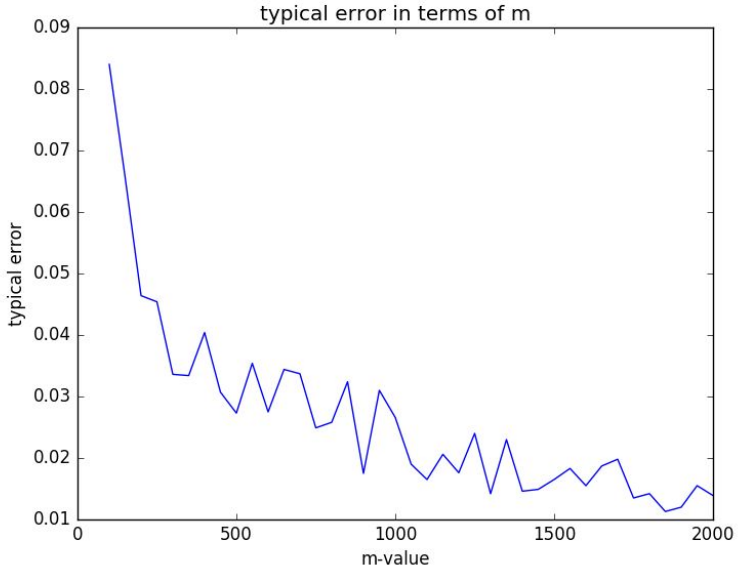
```
# ---------- Question 1 ----------
dataSet, labels = generateDataSet(10000)
tree = buildDecisionTree(dataSet, labels)
print('dataset looks like this:')
print(dataSet)
print(labels)
print('The decision tree by ID3 looks like this:')
print(tree)

print('given decision tree t, calculate error(f):')
print(calculateError(tree, 10000))

print('plot the typical error on m data points as m function:')
m_list = [i for i in range(100, 2050, 50)]
typical_error = [cal_typicalError(m) for m in m_list]
plt.plot(m_list, typical_error)
plt.title('typical error in terms of m')
plt.xlabel('m-value')
plt.ylabel('typical error')
plt.show()
```

```
/Users/huangsuyu/anaconda/bin/python /Users/huangsuyu/Documents/python_project/PrunningDecisionTree.py
dataset looks like this:
[[0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1], [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19', 'X20']
The decision tree by ID3 looks like this:
{'X4': {0: {'X0': {0: {'X2': {0: {'X5': {0: {'X1': {0: 0, 1: {'X3': {0: 0, 1: {'X6': {0: 0, 1: {'X7': {0: 0, 1: 1}}}}}}}}}}}}}, 1: {'X1': {0: {'X3': {0: 0, 1: {'X7': {0: 0, 1: {'X6': {0: 0, 1: 1}}}}}},
given decision tree t, calculate error(f):
0.0083
plot the typical error on m data points as m function:

Process finished with exit code 0
```

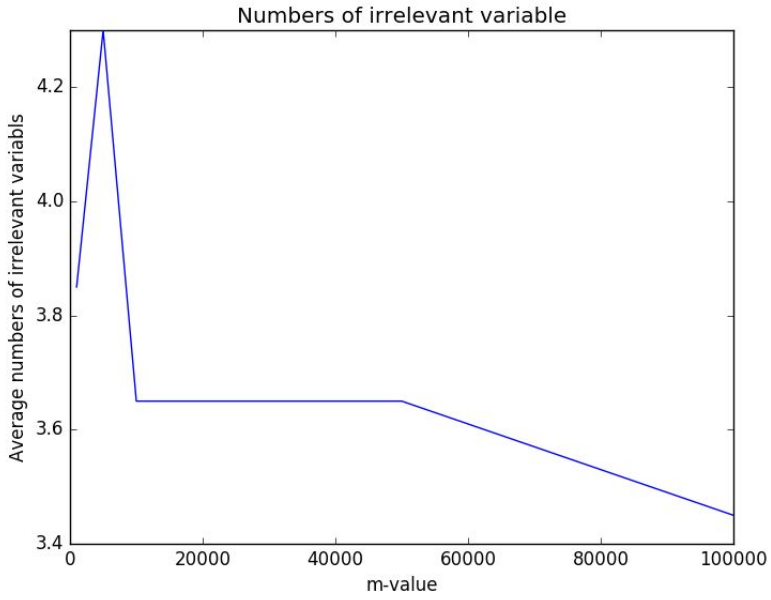



Yes, it agrees with my intuition. As m increases, which means that we have more data, the error rate decreases.

***2) Note that X15 through X20 are completely irrelevant to predicting the value of Y . For a range of m values, repeatedly generate data sets of that size and fit trees to that data, and estimate the average number of irrelevant variables that are included in the fit tree. How much data would you need, typically, to avoid fitting on this noise?***

Solution:

I the average numbers of irrelevant variables as m = 1000, 5000, 10000, 50000, 100000. The result is as below:




The average numbers of noise increases as m becomes large at first. However, when after m = 5000, it decreases as m increases. When it m = 100000, there are 3.45 irrelevant variables approximately.

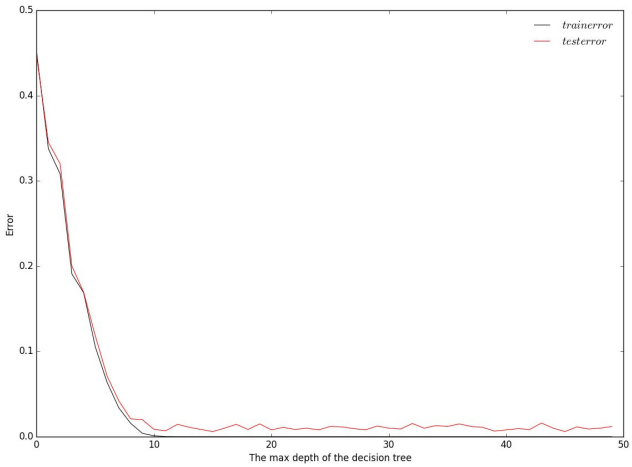
***3) Generate a data set of size m = 10000, and set aside 8000 points for training, and 2000 points for testing. The remaining questions should all be applied to this data set.***

***a) Pruning by Depth: Consider growing a tree as a process - running ID3 for instance until all splits up to depth d have been performed. Depth d = 0 should correspond to no decisions - a prediction for Y is made just on the raw frequencies of Y in the data. Plot, as a function of d, the error on the training set and the error on the test set for a tree grown to depth d. What does your data suggest as a good threshold depth?***

Solution:

The result I plot is as follow:

The data suggest that the depth = 10 approximately is a good threshold depth because when depth reaches to 10, the train error seems to be converged and the test error is equal to 0.

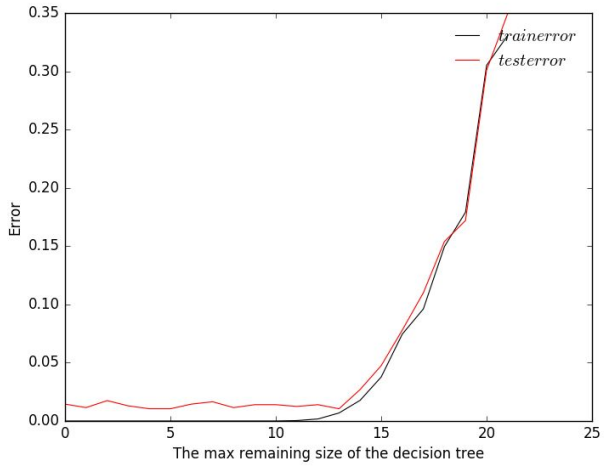



***b) Pruning by Sample Size: The less data a split is performed on, the less 'accurate' we expect the result of that split to be. Let s be a threshold such that if the data available at a node in your decision tree is less than or equal to s, you do not split and instead decide Y by simple majority vote (ties broken by coin flip). Plot, as a function of s, the error on the training set and the error on the testing set for a tree split down to sample size s. What does your data suggest as a good sample size threshold?***

**Solution:**

The result I plot is as follow:

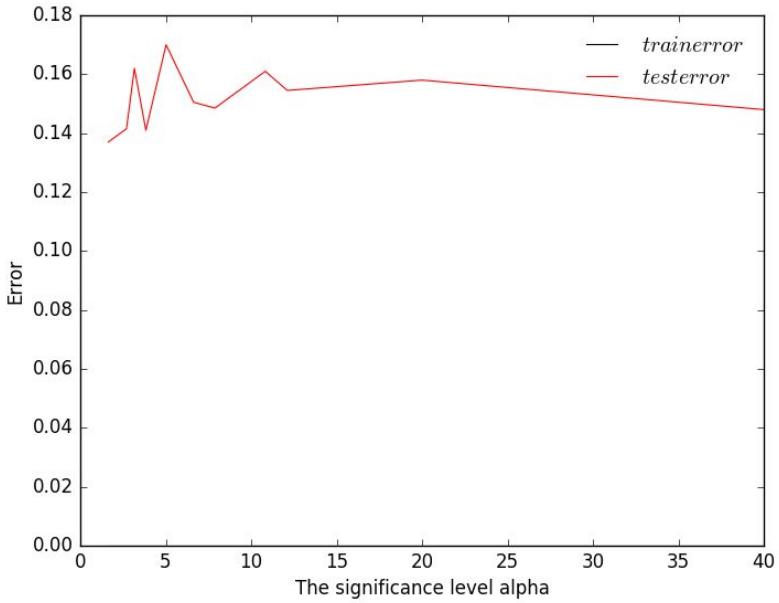
The data suggest that the size = 12 approximately is a good threshold size because when the remaining size becomes 12, the test error seems to be converged as size decreases and the train error goes down to 0.



***c) Pruning by Significance: If a variable X is independent of Y , then X has no value as a splitting variable. We can use something like the χ 2 -test to estimate how likely a potential splitting variable is to be independent, based on the test statistic T compared to some threshold T0 (in the usual 2-outcome case, T0 = 3.841 is used to test at a significance level of p = 5% - see notes for more explanation). Given T0, if given the data for X the value of T is less than T0, it is deemed not significant and is not used for splitting. If given the data for X the value of T is greater than T0, it is deemed significant, and used for splitting. Plot, as a function of T0, the error on the training set and the error on the testing set for a tree split at significance threshold T0. What does your data suggest as a good threshold for significance?***

**Solution:**

The ideal of my program is to calculate the test statistic T for each variables Xi. Then we choose a variable with maximum test statistic T to split dataset. The result is as below:

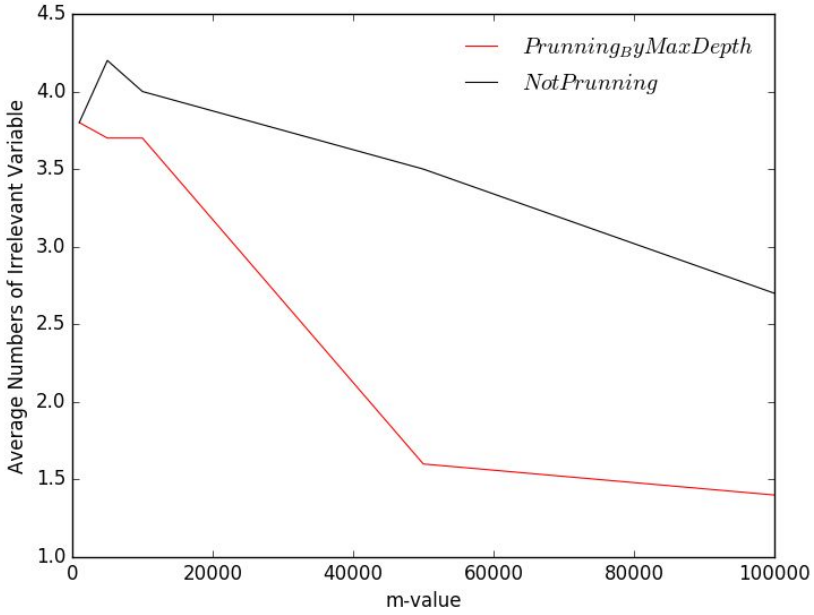



Notice that the train error is 0 all the time in the graph. Because that X0 ~ X14 has very large statistic T value, some of them is greater than 100, when using these variables to split data, the decision tree is being so complicated so that it contains all the observations from training dataset, which leads to the result of having 0 train error.

And because X0 ~ X14 has very large statistic T value, the significance levels play insignificant role in terms of splitting the dataset. Thus the test error shown in the graph seems to be varied only a little as the significance level increases.

***5) Repeat the computation of Problem 2, growing your trees only to depth d as chosen in 3.a. How does this change the likelihood or frequency of including spurious variables in your trees?***

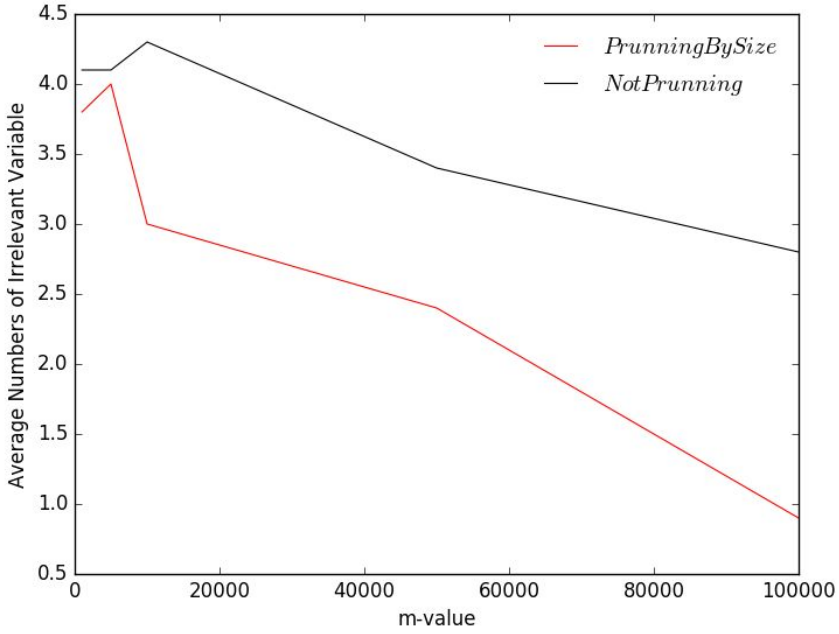
**Solution:**
I run my program with the setting that m is 1000, 5000, 10000, 50000,100000 and max depth to 10. The result is as below:




If we choose the max depth to be 10, the noise can be reduced significantly compared to the one without pruning. As the m increases, the number of irrelevant variables decreases.

***6) Repeat the computation of Problem 2, splitting your trees only to sample size s as chosen in 3.b. How does this change the likelihood or frequency of including spurious variables in your trees?***

**Solution:**
I run my program with the setting that m is 1000, 5000, 10000, 50000,100000 and threshold size to be 12. The result is as below:

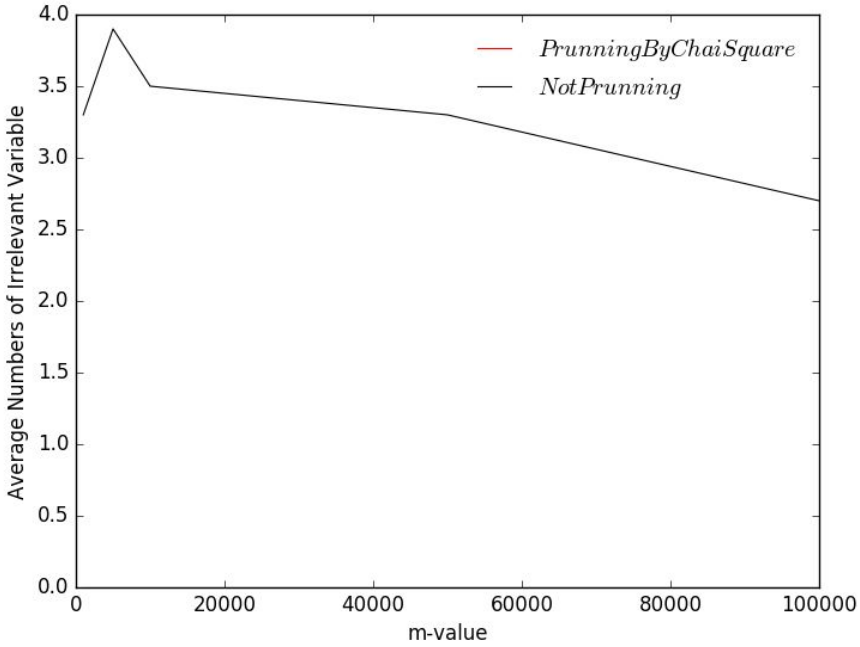



This graph shows that if we choose the size to be 12, the noise can be reduced significantly compared to one without pruning. As the m increases, the number of irrelevant variables decreases.

***7) Repeat the computation of Problem 2, splitting your trees only at or above threshold level T0 as chosen in 3.c. How does this change the likelihood or frequency of including spurious variables in your trees?***

**Solution:**

I run my program with the setting that m is 1000, 5000, 10000, 50000,100000. The result is as below:




This graph shows that when pruning the tree by significance, the noise can be eliminated completely.