Introduction to Data Science

Homework 3

Student Name: Eric Niblock

Student Netid: ejn9259

Part 1 (5 Points)

Assume that X and Y are discrete random variables. The formulas for Mutual Information, Entropy and Conditional Entropy are given by:

$$\text{Mutual Information} = \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot log \frac{p(x,y)}{p(x)p(y)}$$

$$\text{Entropy} = H(Y) = -\sum_{y \in Y} p(y) \cdot log(p(y))$$

$$\text{Conditional Entropy} = H(Y \mid X) = \sum_{x \in X} p(x) \cdot H(Y \mid X = x)$$

Show mathematically that $\text{Mutual Information} = \text{Information Gain}$, where $\text{Information Gain} = H(Y) – H(Y \mid X)$. Give the derivation below (note, this can be done using Latek math notation, which renders nicely. See above. Feel free to do it by hand and submit an image of your proof).

We begin by the definition of mutual information, and move to the definition of information gain,

$$\sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot log \frac{p(x,y)}{p(x)p(y)} = \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot \left(log \frac{p(x,y)}{p(x)} - log(p(y)) \right)$$

$$= \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot log \frac{p(x,y)}{p(x)} - \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot log(p(y))$$

$$= \sum_{y \in Y} \sum_{x \in X} p(x)p(y|x) \cdot log \frac{p(x)p(y|x)}{p(x)} - \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot log(p(y))$$

$$= \sum_{y \in Y} \sum_{x \in X} p(x)p(y|x) \cdot log(p(y|x)) - \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot log(p(y))$$

$$= \sum_{x \in X} p(x) \left(\sum_{y \in Y} p(y|x) \cdot log(p(y|x)) \right) - \sum_{y \in Y} \left(\sum_{x \in X} p(x,y) \right) log(p(y))$$

$$= -\sum_{x \in X} p(x)H(Y|X=x) - \sum_{y \in Y} p(y)log(p(y))$$

$$= H(Y) - H(Y|X)$$

Part 2 - Preparing a Training Set and Training a Decision Tree (10 Points)

This is a hands-on task where we build a predictive model using Decision Trees discussed in class. For this part, we will be using the data in `cell2cell_data.csv` .

These historical data consist of 39,859 customers: 19,901 customers that churned (i.e., left the company) and 19,958 that did not churn (see the `"churndep"` variable). Here are the data set's 11 possible predictor variables for churning behavior:

```
Pos.  Var. Name  Var. Description
----- ---------- ---------------------------------------------------------
-------
1     revenue    Mean monthly revenue in dollars
2     outcalls   Mean number of outbound voice calls
3     incalls    Mean number of inbound voice calls
4     months     Months in Service
5     eqpdays    Number of days the customer has had his/her current equ
ipment
6     webcap     Handset is web capable
7     marryyes   Married (1=Yes; 0=No)
8     travel     Has traveled to non-US country (1=Yes; 0=No)
9     pcown      Owns a personal computer (1=Yes; 0=No)
10    creditcd   Possesses a credit card (1=Yes; 0=No)
11    retcalls   Number of calls previously made to retention team
```

The 12th column, the dependent variable `"churndep"` , equals 1 if the customer churned, and 0 otherwise.

1. Load the data and prepare it for modeling. Note that the features are already processed for you, so the only thing needed here is split the data into training and testing. Use pandas to create two data frames: train_df and test_df, where train_df has 80% of the data chosen uniformly at random without replacement (test_df should have the other 20%). Also, make sure to write your own code to do the splits. You may use any random() function in numpy but DO NOT use the data splitting functions from Sklearn.

In [88]:
```
import pandas as pd
import numpy as np

header_list = ["revenue", "outcalls", "incalls", "months", "eqpdays", "webcap", '
               "travel", "pcown", "creditcd", "retcalls", "churndep"]
data = pd.read_csv("cell2cell_data.csv", names=header_list)

ind = np.arange(len(data))
np.random.shuffle(ind)
train_select = ind[0:int(0.8*len(ind))]
test_select = ind[int(0.8*len(ind)):]

train_df = data.iloc[train_select]
test_df = data.iloc[test_select]
```

2. If we had to, how would we prove to ourselves or a colleague that our data was indeed randomly sampled on X? And by prove, I mean empirically, not just showing this person our code. Don't actually do the work, just describe in your own words a test you could here. Hint: think about this in terms of selection bias and use notes from our 2nd lecture.

In order to effectively prove to a colleague that the data above is void of selection bias, we could start by explaining selection bias. If the splitting of our data was affected by selection bias, this would imply that the probability of finding a specific value corresponding to any feature changes between our sample and when our sample is split (here, we are assuming that the sample itself is void of selection bias). There are a number of possible methods we could employ to show that this is not the case. For example, if we were to run the split method 10,000 times while tracking the placement of a specific record, we should find that on average a record appears in the training data ~80% of the time, and in the test data ~20% of the time. This would help to show that records are being randomly assigned into either training or test buckets, without reference to feature values. Furthermore, after performing 10,000 splits, the mean of any feature within the training data should be approximately equal to the mean of the same feature in the testing data, and also the mean of the feature from the totality of the sample. That method is displayed below, and as is clearly shown, the resulting means are approximately equal.

In [89]:
```
## THIS CODE IS NOT REQUIRED BY THE HOMEWORK AND SIMPLY SHOWS THE RESULTS OF (2)

mean_train = 0
mean_test = 0

header_list = ["revenue", "outcalls", "incalls", "months", "eqpdays", "webcap", '
               "travel", "pcown", "creditcd", "retcalls", "churndep"]
data = pd.read_csv("cell2cell_data.csv", names=header_list)

for t in range(10000):

    ind = np.arange(len(data))
    np.random.shuffle(ind)
    train_select = ind[0:int(0.8*len(ind))]
    test_select = ind[int(0.8*len(ind)):]

    train_df = data.iloc[train_select]
    test_df = data.iloc[test_select]
    mean_train += np.mean(train_df['revenue'])
    mean_test += np.mean(test_df['revenue'])
print('Mean Revenue (10000 trails) Training Data: ', mean_train/10000)
print('Mean Revenue (10000 trails) Test Data: ', mean_test/10000)
print('Mean Revenue Actual (sample): ', np.mean(data['revenue']))
```

```
Mean Revenue (10000 trails) Training Data:  58.63412180415116
Mean Revenue (10000 trails) Test Data:  58.632593832287846
Mean Revenue Actual (sample):  58.63381620211279
```

3. Now build and train a decision tree classifier using `DecisionTreeClassifier()` [(manual page) (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) on train_df to predict the `"churndep"` target variable. Make sure to use `criterion='entropy'` when instantiating an instance of `DecisionTreeClassifier()`. For all other settings you should use all of the default options.

In [90]:
```
from sklearn import tree
import matplotlib.pyplot as plt

train_X = train_df.drop(['churndep'], axis=1)
train_Y = train_df['churndep']

clf = tree.DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(train_X, train_Y)
```

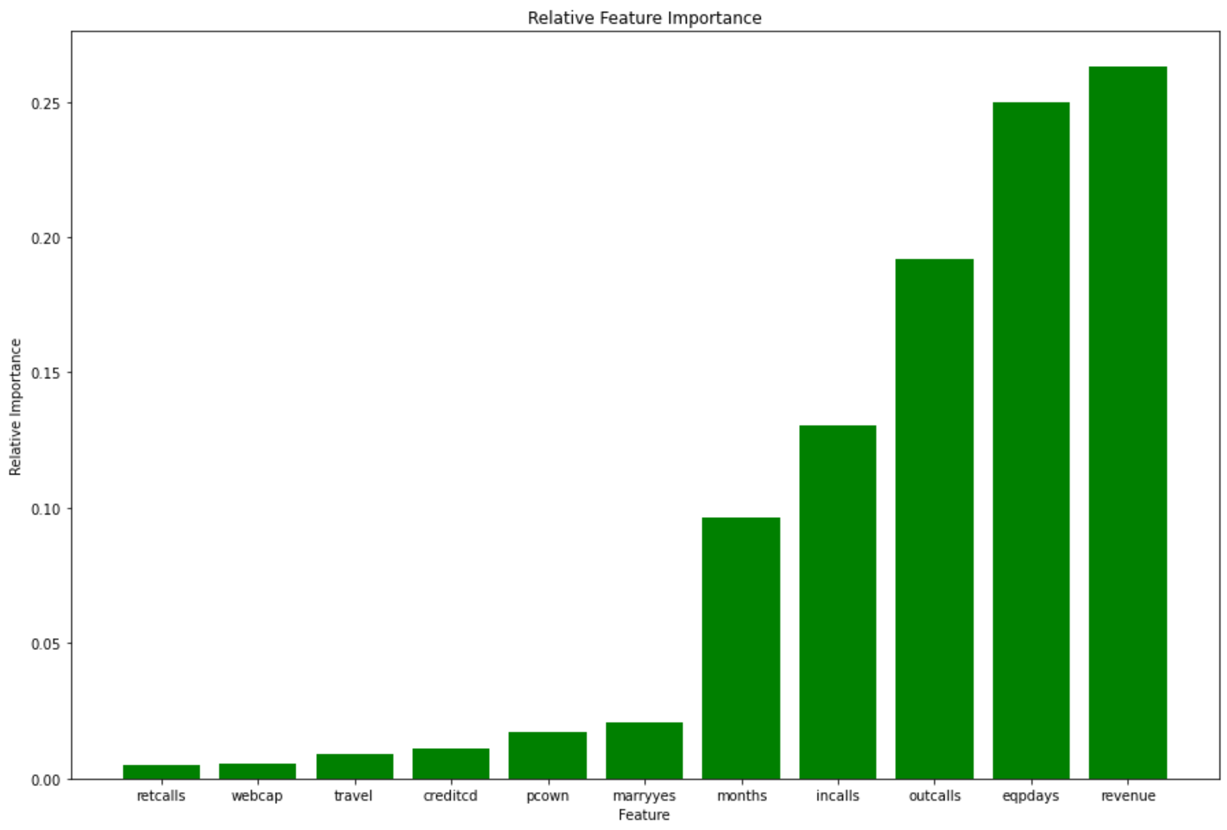
4. Using the resulting model from 2.3, show a bar plot of feature names and their feature importance (hint: check the attributes of the `DecisionTreeClassifier()` object directly in IPython or check the manual!). Make sure the bar plot is sorted by increasing feature importance values.

In [91]:
```
%matplotlib inline

importance = list(clf.feature_importances_)
labels = list(data.columns)
importance, labels = zip(*sorted(zip(importance, labels)))

plt.figure(figsize=(15,10))
plt.bar(labels, importance, color='green')
plt.xlabel("Feature")
plt.ylabel("Relative Importance")
plt.title("Relative Feature Importance")
```

Out[91]:
```
Text(0.5, 1.0, 'Relative Feature Importance')
```



5. Is the relationship between the top 3 most important features (as measured here) negative or positive? If your marketing director asked you to explain the top 3 drivers of churn, how would you interpret the relationship between these 3 features and the churn outcome? What "real-life" connection can you draw between each variable and churn? Make sure to state your answer, and not just show code.

In [92]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

print(stats.pointbiserialr(train_df['revenue'],train_df['churndep']))
print(stats.pointbiserialr(train_df['eqpdays'],train_df['churndep']))
print(stats.pointbiserialr(train_df['outcalls'],train_df['churndep']))
```

```
PointbiserialrResult(correlation=-0.01315213458110532, pvalue=0.018845294419533
56)
PointbiserialrResult(correlation=0.10870353708289446, pvalue=2.0486264798571453
e-84)
PointbiserialrResult(correlation=-0.03587104671292903, pvalue=1.481395878601493
e-10)
```

Given the correlation values between each of the most important features and the value of the churn variable, we find that revenue and churn are negatively correlated, which makes sense given that customers which are paying more are more likely to be burdened by high costs, and more willing to give up on a product if it will save them significant amounts of money (they might switch to a competitor). We also find that the number of days someone has had their equipment and churn are positively correlated, which makes sense given that customers who have had their equipment for a long time are more likely to be dissatisfied with a product's performance because it is becoming old and out of date. Finally, the mean number of outbound calls and the rate of churn are negatively correlated, which makes sense because customers who are dissatisfied with a product are less likely to utilize it.

6. Using the classifier built in 2.3, try predicting "churndep" on both the train_df and test_df data sets. What is the accuracy on each? What is your explanation on the difference (or lackthereof) between the two accuracies?

In [93]:
```
test_X = test_df.drop(['churndep'], axis=1)
test_Y = test_df['churndep']

print('Training Accuracy: ' + str(clf.score(train_X, train_Y)))
print('Testing Accuracy: ' + str(clf.score(test_X, test_Y)))
```

```
Training Accuracy: 0.9998431962868881
Testing Accuracy: 0.5296036126442549
```

The classifier has been allowed too much freedom, and has over fit the training data. Essentially, the model has memorized the training data, producing very high variance, and low generalizability. Hence, when deployed on the testing data, the model's inflexibility shows, and the variability that exists between the training set and the testing set cannot be handled by the classifier.

Part 3 - Finding a Good Decision Tree (10 Points)

The default options for your decision tree may not be optimal. We need to analyze whether tuning the parameters can improve the accuracy of the classifier. For the following options `min_samples_split` and `min_samples_leaf` :

1. Generate a list of 10 values of each for the parameters min_samples_split and min_samples_leaf.

In [94]:
```
min_samples_split_values = [2,20,70,100,300,500,700,1000,2000,10000]
min_samples_leaf_values = [2,20,70,100,300,500,700,1000,2000,10000]
```

2. Explain in words your reasoning for choosing the above ranges.

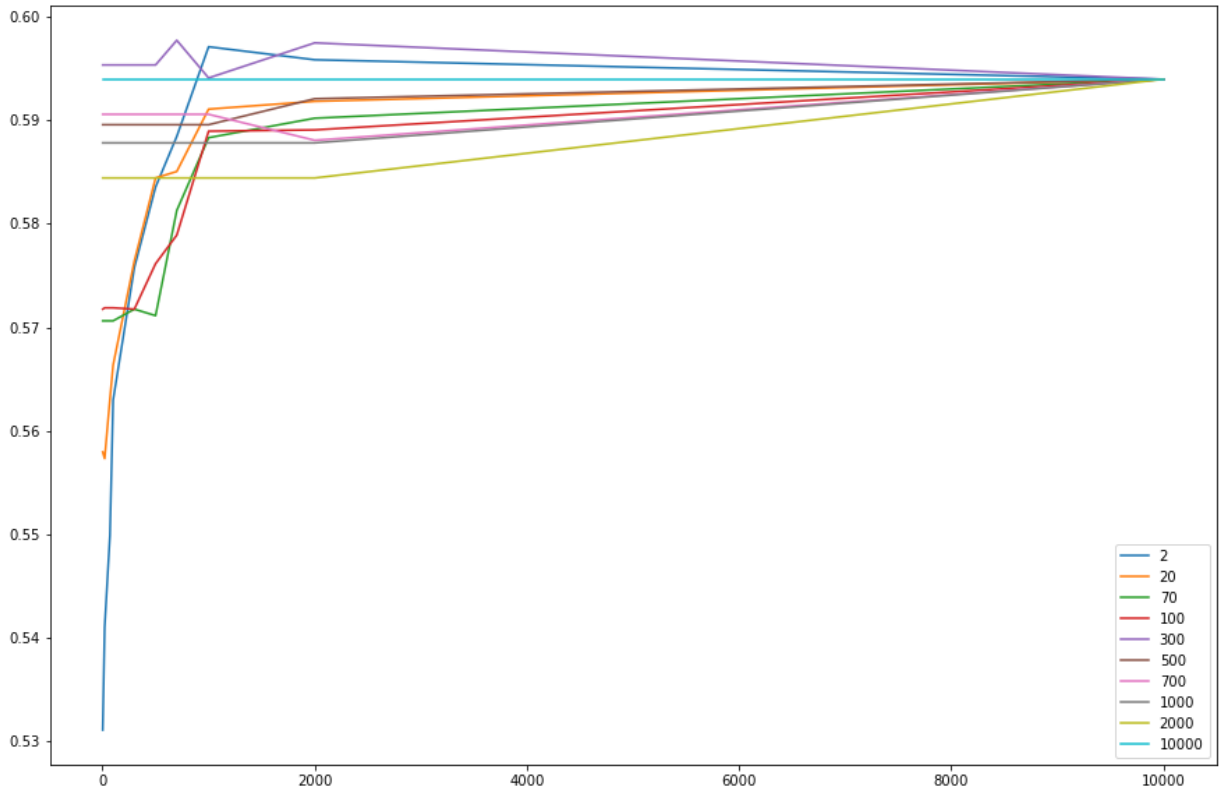
Though this process is very much so dependent on a trial-and-error type of approach, an understanding of min_samples_split and min_samples_leaf helped to inform my decision. Without adjustment, tree.DecisionTreeClassifier uses a min_samples_split value of 2, implying that a new branch of the decision tree can be produced provided that at least two samples remain to be split. This is not very productive, because it encourages the classifier to split samples down to a very fine level of granularity (with min_samples_leaf defaulting to one, the classifier is encouraged to continue splitting until each sample has its own branch, ending in a leaf). These default parameters encouraged overfitting. To correct for these, the above ranges were chosen because, given that our target variable is binary, it seems likely that a larger value concerning min_samples_split and min_samples_leaf would prevent the loss of generalizability associated with the splitting of the data into leafs that only possess one data point. We know that the data contains around 20,000 records corresponding to each value of our target variable, and as such we know that an absolute upper bound for min_samples_leaf should be 20,000 (otherwise a split would not be allowed). Furthermore, we know this value is still too large, because we want to be able to obtain a reasonable level of depth (perhaps at least a depth level of 2, which would correspond to roughly 10,000). A very similar line of thought is applied to min_samples_split.

3. For each combination of values in 3.1 (there should be 100), build a new classifier and check the classifier's accuracy on the test data. Plot the test set accuracy for these options. Use the values of `min_samples_split` as the x-axis and generate a new series (line) for each of `min_samples_leaf`.

In [95]:
```
import matplotlib.pyplot as plt
%matplotlib inline

series = []
for leaf in min_samples_leaf_values:
    splits_vals = []
    for split in min_samples_split_values:
        clf = tree.DecisionTreeClassifier(criterion='entropy', min_samples_split=
        clf = clf.fit(train_X, train_Y)
        score = clf.score(test_X, test_Y)
        splits_vals.append(score)
    series.append(splits_vals)
```

In [96]:
```
plt.figure(figsize=(15,10))
for l in range(len(series)):
    plt.plot(min_samples_split_values, series[l], label=str(min_samples_leaf_valu
    plt.legend()
```



4. Which configuration returns the best accuracy? What is this accuracy? (Note, if you don't see much variation in the test set accuracy across values of min_samples_split or min_samples_leaf, try redoing the above steps with a different range of values), and reassess your answer in Q3.2.

In [101]:
```
largest = 0
for leaf in range(len(min_samples_leaf_values)):
    for split in range(len(min_samples_split_values)):
        if series[leaf][split] > largest:
            indexs = [leaf, split]
            largest = series[leaf][split]
print('Best leaf and split vals: ', min_samples_leaf_values[indexs[0]], min_sampl
print('Test Accuracy: ', series[indexs[0]][indexs[1]])
```

```
Best leaf and split vals:  300 700
Test Accuracy:  0.5977170095333668
```

5. If you were working for a marketing department, how would you use your churn production model in a real business environment? Explain why churn prediction might be good for the business and how one might improve churn by using this model.

Churn prediction models are important in business for a number of reasons. Having a model such as this, a sales and marketing division could regularly assess their customer database and predict which users, subscribers, customers, etc. are at risk for churning. Once this information is obtained, customers at a high risk for churn could be provided with incentives in order to stay - this could mean personalized outreach from someone within the marketing department (or another company representative), or even discounts and special offers. Further research could be done into the specific risk factors that are highly correlated to churn in order to provide personalized offers to customers (if 'revenue' seems to be the predominant factor involved, offer some financial discount/incentive, if 'eqpdays' is driving the classifier, offer a discount on upgrading to better equipment). Overall, the model could represent a valuable tool in increasing revenue for the company, by persuading high risk customers to stay by evaluating and assessing the factors which made them a high risk for churning in the first place. This would also serve to decrease future churn.

In [103]:
```
## CODE NOT REQUIRED - OPTIMAL SOLUTION FOR FINDING BEST COMBINATION OF LEAF AND
## THE BEST METHOD WOULD PROBABLY BE ONE THAT TESTS VALUES AND THEN HONES IN ON A

min_samples_split_values = [i*20 for i in range(1,20)]
min_samples_leaf_values = [i*25 for i in range(1,50)]

series = []
for leaf in min_samples_leaf_values:
    splits_vals = []
    for split in min_samples_split_values:
        clf = tree.DecisionTreeClassifier(criterion='entropy', min_samples_split=
        clf = clf.fit(train_X, train_Y)
        score = clf.score(test_X, test_Y)
        splits_vals.append(score)
    series.append(splits_vals)

largest = 0
for leaf in range(len(min_samples_leaf_values)):
    for split in range(len(min_samples_split_values)):
        if series[leaf][split] > largest:
            indexs = [leaf, split]
            largest = series[leaf][split]
print(min_samples_leaf_values[indexs[0]], min_samples_split_values[indexs[1]])
series[indexs[0]][indexs[1]]
```

```
375 20
```

Out[103]:
```
0.5963371801304566
```

In [ ]: