Luis Castaneda - lcasta6@uic.edu
David Hernandez - dherna71@uic.edu

PART I

Project Description

Our Final Project consisted of predicting the salary of a given individual considering features presented by them by the “adult.data” file found in the UCI machine learning repository. This data file consisted of a combination of string and integers used as features for the label (i.e whether the individual made +50k+ or not per year).

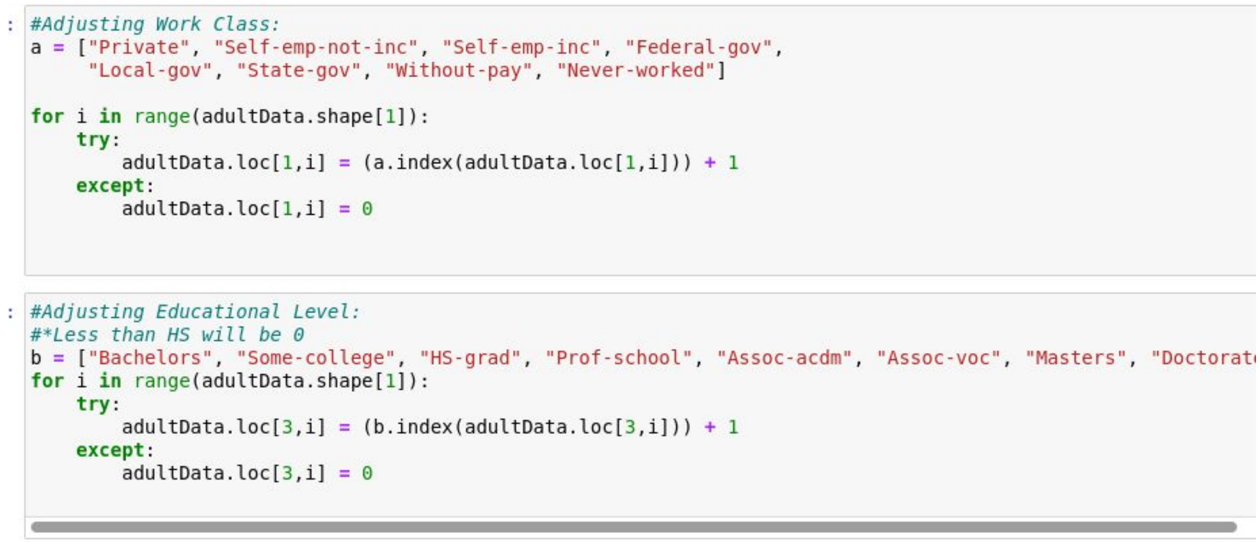
Data being used: https://archive.ics.uci.edu/ml/datasets/Adult

Complications with the data file

Given the above, we were forced to use the pandas dataframe found in the pandas library instead of the numpy array that we’ve been accustomed to in this course. This was because numpy arrays were not able to be adequately matched to hold strings and integers at the same time. A combination of both types of features could best have been done with the pandas dataFrame.

Processing the data

After the data was held we attempted to use our classifiers as normal, but found that the first classifier that we tried (KNN) was unable to process strings in it's ML process. For that reason we decided to convert all our strings into integer weights within the data frame. The integers ranged from 0 - N, where N is the number of possible strings for that given category of labels. Any strings outside of these classifications, like missing data where handled by giving them a value of 0. This presented a unique set of challenges as we were encountering this problem that was not a factor in this course, as well as working with a new data structure different from what we had been using.



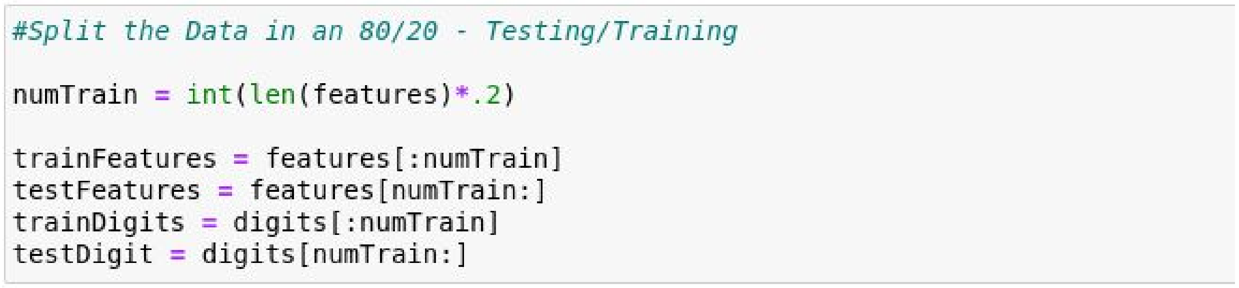
```
: #Adjusting Work Class:
a = ["Private", "Self-emp-not-inc", "Self-emp-inc", "Federal-gov",
     "Local-gov", "State-gov", "Without-pay", "Never-worked"]

for i in range(adultData.shape[1]):
    try:
        adultData.loc[1,i] = (a.index(adultData.loc[1,i])) + 1
    except:
        adultData.loc[1,i] = 0
```

```
: #Adjusting Educational Level:
#*Less than HS will be 0
b = ["Bachelors", "Some-college", "HS-grad", "Prof-school", "Assoc-acdm", "Assoc-voc", "Masters", "Doctorat
for i in range(adultData.shape[1]):
    try:
        adultData.loc[3,i] = (b.index(adultData.loc[3,i])) + 1
    except:
        adultData.loc[3,i] = 0
```

Separating the Data

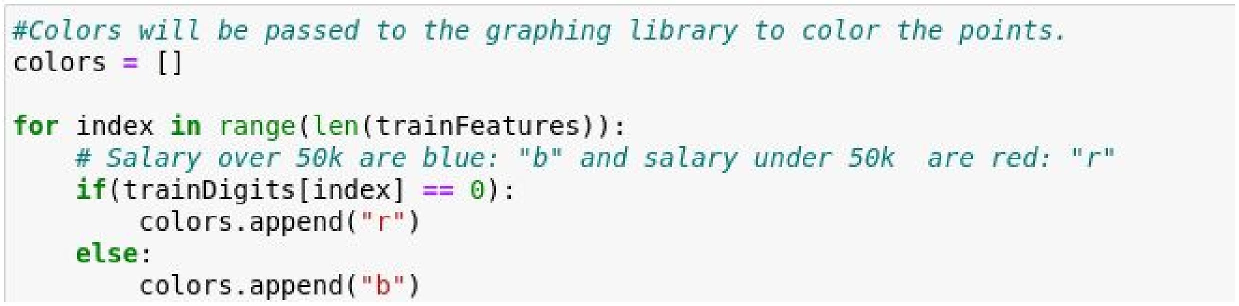
After the above was completed, now we had to seperate our data. We decided that an 80/20 testing to training separation would work just fine for our purposes. We also took this time to prepare color separation for our first analysis of the data.(Kernel PCA)



```
#Split the Data in an 80/20 - Testing/Training

numTrain = int(len(features)*.2)

trainFeatures = features[:numTrain]
testFeatures = features[numTrain:]
trainDigits = digits[:numTrain]
testDigit = digits[numTrain:]
```

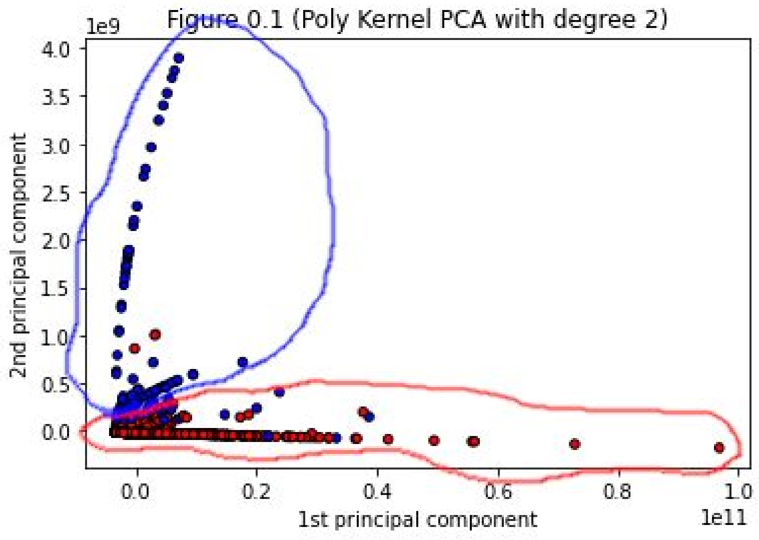


```
#Colors will be passed to the graphing library to color the points.
colors = []

for index in range(len(trainFeatures)):
    # Salary over 50k are blue: "b" and salary under 50k  are red: "r"
    if(trainDigits[index] == 0):
        colors.append("r")
    else:
        colors.append("b")
```

Project Description

We conducted an analysis of the data using KPCA to try and determine whether people who make more than 50k and less than 50k had features similar to each other. Our prediction is that there would be a distinction between people that meet the threshold and people that didn't. We hypothesised that factors such as education, age, and marital status would be a good indication of the level of income from one individual to another. What factors contributed more to our KPCA is difficult to say, but according to our figure we were roughly correct. The data does show that people that made more than 50k (red dots) did cluster within each other more frequently than they did with those that make less than 50k. It also shows that there were other features that played a minor, or unessential role, in determining our result; this can be seen with the blue and red dot clusters being proximal to each other, while at the same time clearly separated. (Figure 0.1)



Conclusions from the KPCA

The KPCA proved that there were factors that could be used to determine the income of certain individuals. While we did see some overlap the separation was also pretty clear between the two clusters. This was a good data set for analysis.

KNN Classifier

It was predicted that this classifier would also be the most accurate out of all the classifiers that we would attempt. This turned out to not have been the case as there would be more accurate classifiers that we would encounter later on.

From our figure (**Figure 1.1**) we were correct in predicting that as the number of neighbors being considered increased, so did the accuracy of our mode. The result of accuracy of the model did start to train off after 10 nearest neighbors; this shows that there was indeed a threshold from which we would see diminishing returns. The KNN was also tested with different distance parameters (Chebyshev[Figure 1.2] , Manhattan[Figure 1.3]) but they made no noticeable difference. KNN gave us a peak accuracy of about 80%. KNN using Euclidean distance gave us a score of 79% for the test data.

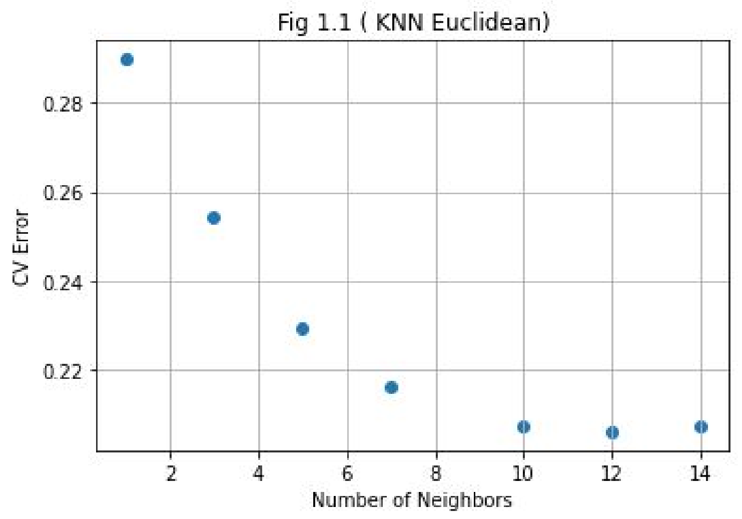


Fig 1.2 (KNN Manhattan)

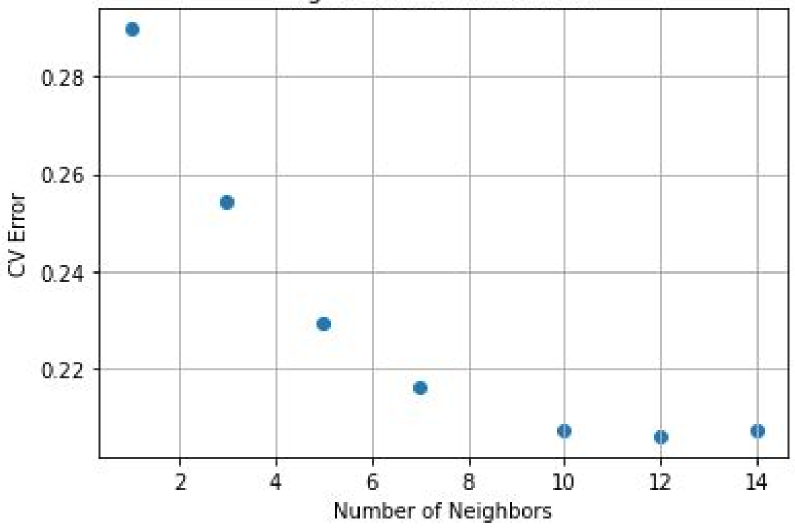
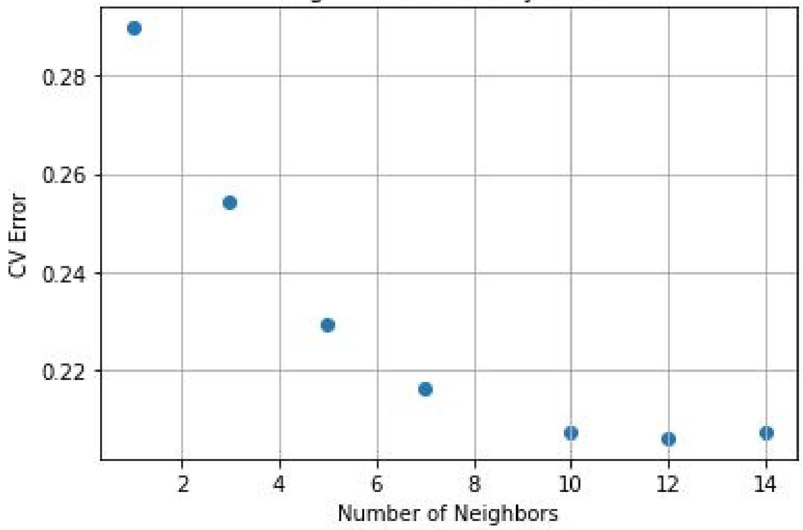


Fig 1.3 (KNN Chebyshev)

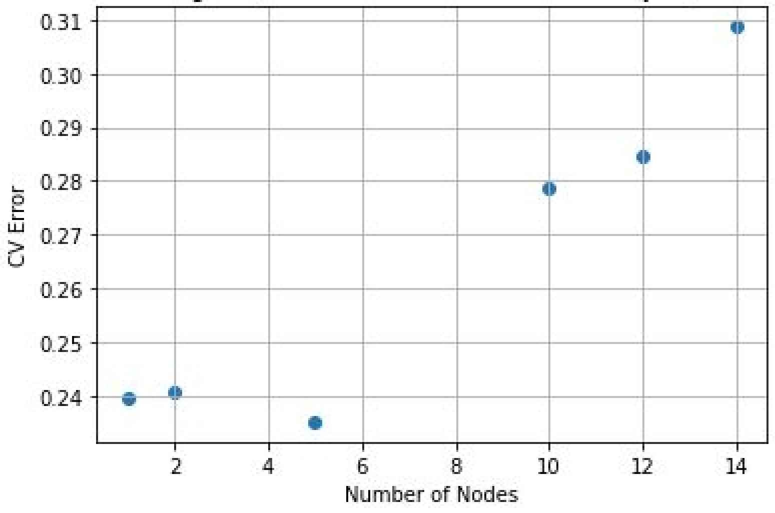


Neural Network

Our next Classifier was the Neural Network, given it’s reputation as a great ML classifier, we were also sure that this would be at the top of our list for the more accurate models. In our case we used the Neural Network with a combination of different hidden layers and number of nodes per layer. The hidden layers ranged where [1,2,5,7,9] and the number of nodes per layer were [1,2,5,10,12,14]. Our results are all shown below. Our most accurate combination ended up being two hidden layers with two nodes giving us an accuracy of 78%. This would again not be as accurate as our other models. Using the best optimal model for the Neural network it got a score of 78% when using the test data.

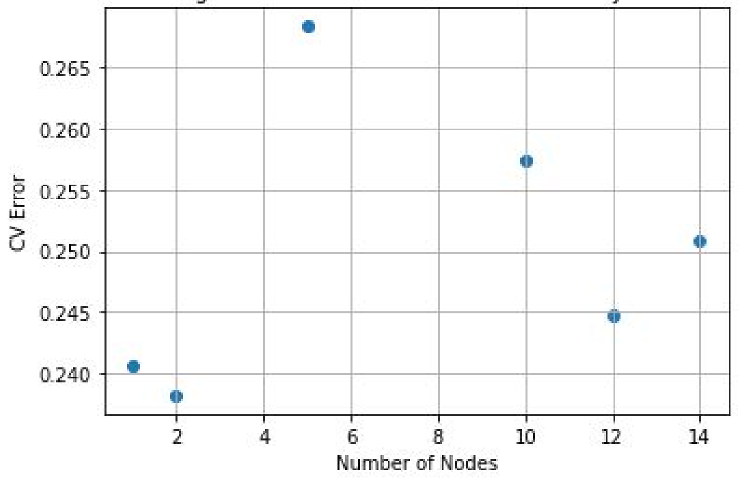
```
Accuracy : (Layers = 1 | Nodes = 1) : 0.76 (+/- 0.01)
Accuracy : (Layers = 1 | Nodes = 2) : 0.76 (+/- 0.00)
Accuracy : (Layers = 1 | Nodes = 5) : 0.76 (+/- 0.03)
Accuracy : (Layers = 1 | Nodes = 10) : 0.72 (+/- 0.22)
Accuracy : (Layers = 1 | Nodes = 12) : 0.72 (+/- 0.27)
Accuracy : (Layers = 1 | Nodes = 14) : 0.69 (+/- 0.32)
```

Fig 2.1 (Neural Network One Hidden Layer)



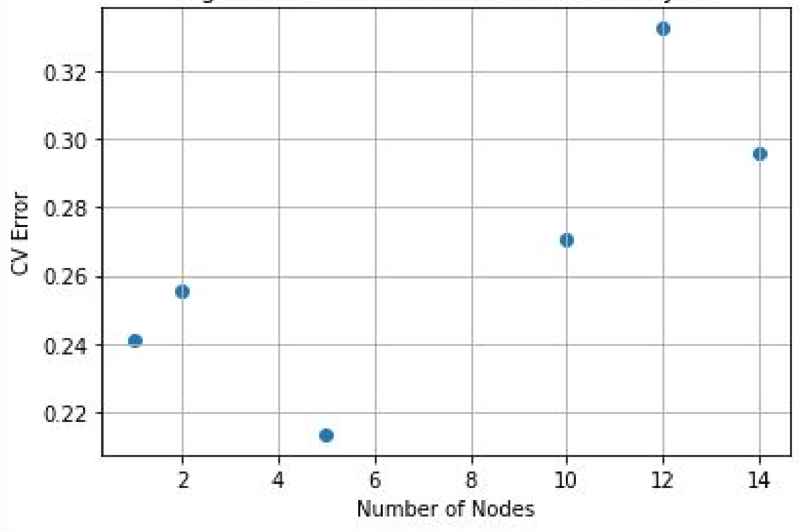
```
Accuracy : (Layers = 2 | Nodes = 1) : 0.76 (+/- 0.00)
Accuracy : (Layers = 2 | Nodes = 2) : 0.76 (+/- 0.02)
Accuracy : (Layers = 2 | Nodes = 5) : 0.73 (+/- 0.19)
Accuracy : (Layers = 2 | Nodes = 10) : 0.74 (+/- 0.14)
Accuracy : (Layers = 2 | Nodes = 12) : 0.76 (+/- 0.03)
Accuracy : (Layers = 2 | Nodes = 14) : 0.75 (+/- 0.12)
```

Fig 2.2 (Neural Network Two Hidden Layer)



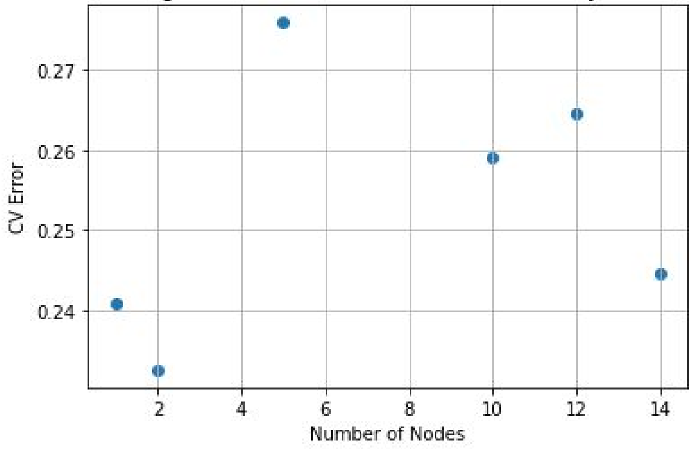
```
Accuracy : (Layers = 5 | Nodes = 1) : 0.76 (+/- 0.00)
Accuracy : (Layers = 5 | Nodes = 2) : 0.74 (+/- 0.13)
Accuracy : (Layers = 5 | Nodes = 5) : 0.79 (+/- 0.04)
Accuracy : (Layers = 5 | Nodes = 10) : 0.73 (+/- 0.25)
Accuracy : (Layers = 5 | Nodes = 12) : 0.67 (+/- 0.42)
Accuracy : (Layers = 5 | Nodes = 14) : 0.70 (+/- 0.31)
```

Fig 2.3 (Neural Network Five Hidden Layer)



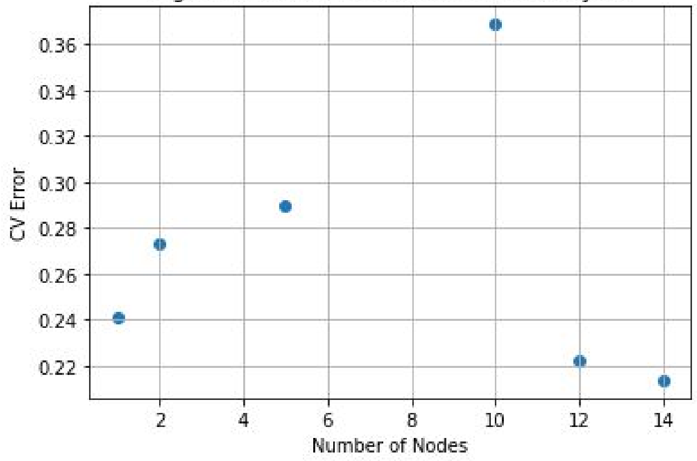
```
Accuracy : (Layers = 7 | Nodes = 1) : 0.76 (+/- 0.00)
Accuracy : (Layers = 7 | Nodes = 2) : 0.77 (+/- 0.03)
Accuracy : (Layers = 7 | Nodes = 5) : 0.72 (+/- 0.31)
Accuracy : (Layers = 7 | Nodes = 10) : 0.74 (+/- 0.19)
Accuracy : (Layers = 7 | Nodes = 12) : 0.74 (+/- 0.21)
Accuracy : (Layers = 7 | Nodes = 14) : 0.76 (+/- 0.22)
```

Fig 2.4 (Neural Network Seven Hidden Layer)



```
Accuracy : (Layers = 9 | Nodes = 1) : 0.76 (+/- 0.00)
Accuracy : (Layers = 9 | Nodes = 2) : 0.73 (+/- 0.28)
Accuracy : (Layers = 9 | Nodes = 5) : 0.71 (+/- 0.25)
Accuracy : (Layers = 9 | Nodes = 10) : 0.63 (+/- 0.41)
Accuracy : (Layers = 9 | Nodes = 12) : 0.78 (+/- 0.08)
Accuracy : (Layers = 9 | Nodes = 14) : 0.79 (+/- 0.03)
```

Fig 2.5 (Neural Network Nine Hidden Layer)

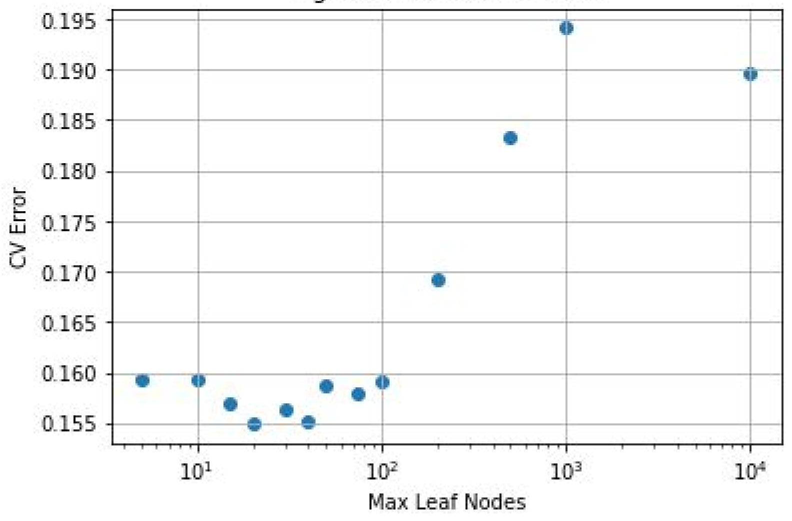


Decision Tree

For our Decision Tree, we decided to use only the number of leafs as the primary differentiating factor between one figure and the other. The Decision tree would turn out to be one of our more accurate models, contradicting our initial suspicions; Our most accurate parameter turned out to be a max leaf value of 20, giving us an accuracy of 85 %.[Figure 3.1]. When getting the score using the test data it got 85%.

```
Accuracy : (Max Leaf Nodes = 5) : 0.84 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 10) : 0.84 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 15) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 20) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 30) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 40) : 0.84 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 50) : 0.84 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 75) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 200) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 500) : 0.82 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000) : 0.81 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10000) : 0.81 (+/- 0.02)
```

Figure 3.1 (Decision Tree)

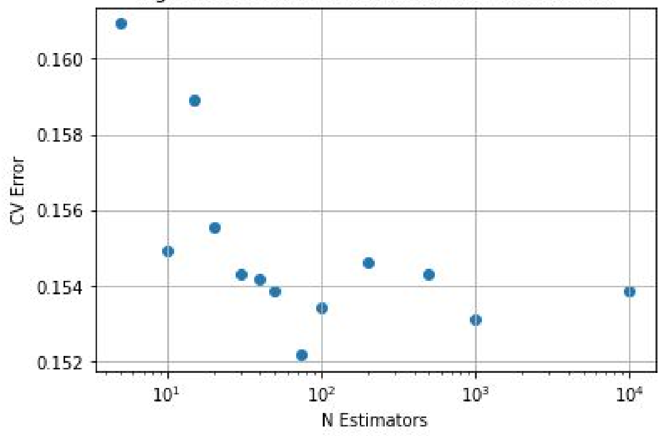


Random Forest

After our decision tree, the most natural procession was to move on to Random Forest. The random Forest was tested with a max_leaf count of [10,100,1000], each with an n_estimator value of [5, 10, 15, 20, 30, 40, 50, 75, 100, 200, 500, 1000, 10000]. This would end giving us our most accurate amount of 86%. The optimal model for Random Forest Tree is 100 Max Leaf Nodes and 75 N-estimators. Any more Max Leaf Nodes does not improve accuracy. Same goes for N-Estimators. When getting a score using test data it got 85% accuracy.

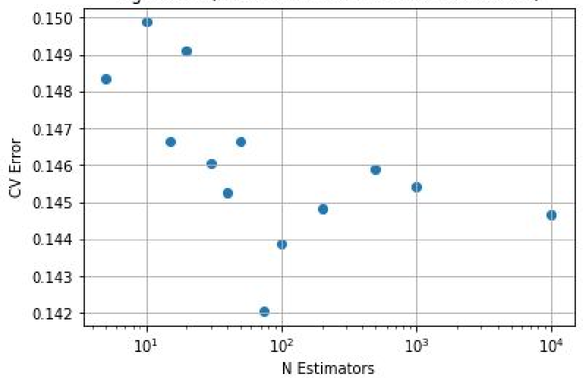
```
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  5) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  10) : 0.85 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  15) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  20) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  30) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  40) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  50) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  75) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  100) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  200) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  500) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  1000) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10 | N-Estimators =  10000) : 0.85 (+/- 0.02)
```

Figure 4.1 (Random Forest 10 Max Leaf Nodes)



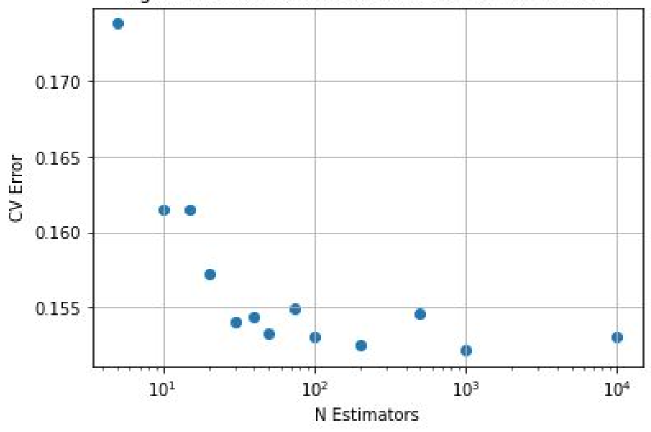
```
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  5) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  10) : 0.85 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  15) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  20) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  30) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  40) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  50) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  75) : 0.86 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  100) : 0.86 (+/- 0.03)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  200) : 0.86 (+/- 0.03)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  500) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  1000) : 0.85 (+/- 0.03)
Accuracy : (Max Leaf Nodes = 100 | N-Estimators =  10000) : 0.86 (+/- 0.03)
```

Figure 4.2 (Random Forest 100 Max Leaf Nodes)



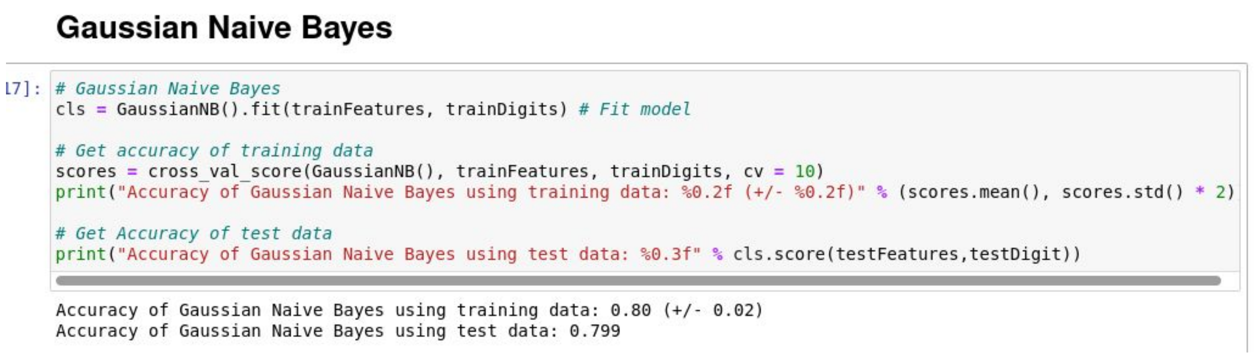
```
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  5) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  10) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  15) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  20) : 0.84 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  30) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  40) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  50) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  75) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  100) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  200) : 0.85 (+/- 0.03)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  500) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  1000) : 0.85 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000 | N-Estimators =  10000) : 0.85 (+/- 0.02)
```

Figure 4.3 (Random Forest 1000 Max Leaf Nodes)



Gaussian Naive Bayes

Our final classifier was the Gaussian Naive Bayes with an accuracy of about 80%. As the name implies, this classifier does not have any tuning parameters so there wasn't anything to tune. It was only plug and play. Yet it got a score of 80% using the training data and 79% using the test data.



Gaussian Naive Bayes

```
[7]: # Gaussian Naive Bayes
cls = GaussianNB().fit(trainFeatures, trainDigits) # Fit model

# Get accuracy of training data
scores = cross_val_score(GaussianNB(), trainFeatures, trainDigits, cv = 10)
print("Accuracy of Gaussian Naive Bayes using training data: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2)

# Get Accuracy of test data
print("Accuracy of Gaussian Naive Bayes using test data: %0.3f" % cls.score(testFeatures,testDigit))
```

```
Accuracy of Gaussian Naive Bayes using training data: 0.80 (+/- 0.02)
Accuracy of Gaussian Naive Bayes using test data: 0.799
```

Citations for Gaussian Naive Bayes
https://www.youtube.com/watch?v=uHK1-Q8cKAw
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
https://kraj3.com.np/blog/2019/07/implementataion-of-naive-bayes-in-pythonusing-scikit-learn/

Conclusion

After testing various optimal models, the best model is Random Forest. Random Forest gave the most consistent highest accuracy of around 85% using training data and the test data. This data set does have features that have a strong relationship with the output. The question is which features have the most impact with the output.

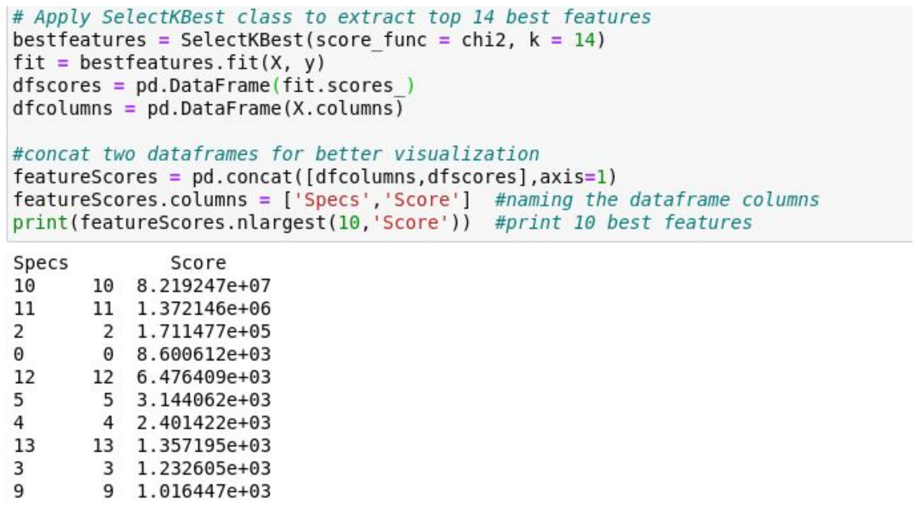
PART II

Part II Description

The second part of our project consisted of us finding the most predictive features out of our available options. We used the SelectKBest and Extra Trees from the sklearn to find which one of these features were the most significant when determining the salary in our data. Out of the 14 features we found that the most predictive feature was Capital Gain [10] first and Capital Loss [11] second according to SelectKBest classifier. This tells us that the individuals with a 50k+ salary are also the ones that have investments in non-liquid assets.

It could also mean that only those individuals with a 50k+ salary were the only ones able to invest in these non-liquid assets. As opposed to someone with a sub 50k salary that does not have as many funds to invest. Regardless we decided to use these two features for further analysis.

K-Best Features Score



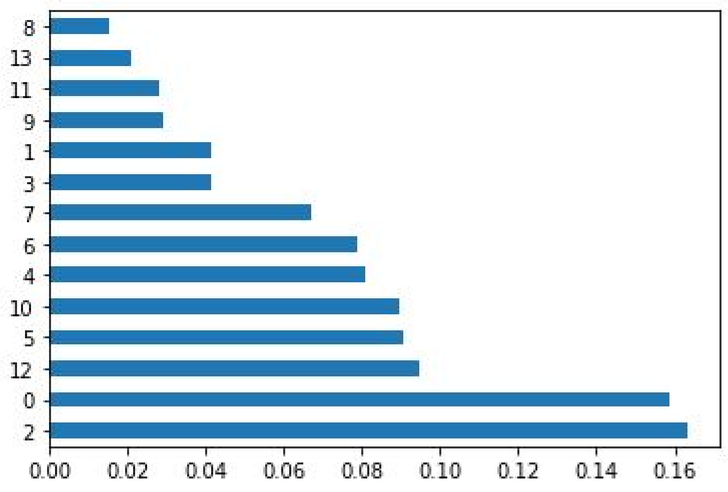
```
# Apply SelectKBest class to extract top 14 best features
bestfeatures = SelectKBest(score_func = chi2, k = 14)
fit = bestfeatures.fit(X, y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
Specs          Score
10      10  8.219247e+07
11      11  1.372146e+06
2        2  1.711477e+05
0        0  8.600612e+03
12      12  6.476409e+03
5        5  3.144062e+03
4        4  2.401422e+03
13      13  1.357195e+03
3        3  1.232605e+03
9        9  1.016447e+03
```

Feature List:
0 - Age
1 - Work Class
2 - Final Weight
3 - Education
4 - Education Num
5 - Marital Status
6 - Occcupation
7 - Relationship
8 - Race
9 - Sex
10 - Capital Gain
11 - Capital Loss
12 - Hours per Week
13 - Native Country

Extra Trees Features Score



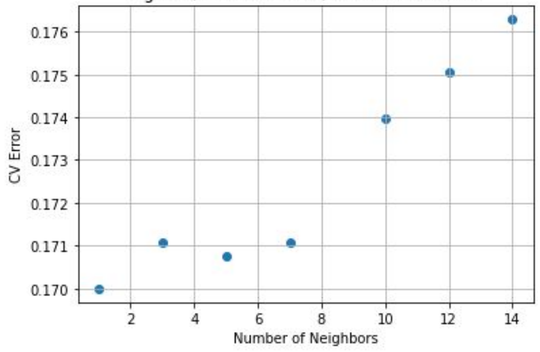
Feature List:
0 - Age
1 - Work Class
2 - Final Weight
3 - Education
4 - Education Num
5 - Marital Status
6 - Occcupation
7 - Relationship
8 - Race
9 - Sex
10 - Capital Gain
11 - Capital Loss
12 - Hours per Week
13 - Native Country

KNN (K-Best)

To demonstrate the difference in accuracy between best predictive features, and our worst predictive features we decided to train a KNN classifier only considering the best [Figure 1.1] and worst features [Figure 1.2] according to K-Best scores. As we see there is a difference of close to 10% between the best determinants and the worst.

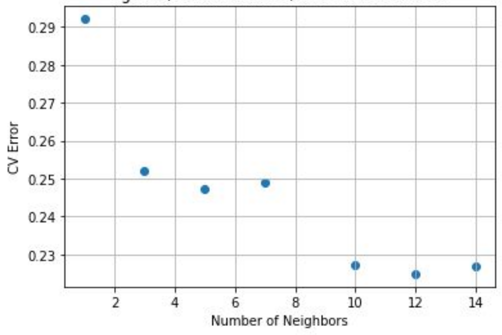
```
Accuracy : (Distance = Euclidean | Neighbors = 1) : 0.83 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 3) : 0.83 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 5) : 0.83 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 7) : 0.83 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 10) : 0.83 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 12) : 0.82 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 14) : 0.82 (+/- 0.02)
```

Fig 1.1 (KNN Euclidean) Best Two Features



```
Accuracy : (Distance = Euclidean | Neighbors = 1) : 0.71 (+/- 0.09)
Accuracy : (Distance = Euclidean | Neighbors = 3) : 0.75 (+/- 0.07)
Accuracy : (Distance = Euclidean | Neighbors = 5) : 0.75 (+/- 0.06)
Accuracy : (Distance = Euclidean | Neighbors = 7) : 0.75 (+/- 0.06)
Accuracy : (Distance = Euclidean | Neighbors = 10) : 0.77 (+/- 0.03)
Accuracy : (Distance = Euclidean | Neighbors = 12) : 0.78 (+/- 0.03)
Accuracy : (Distance = Euclidean | Neighbors = 14) : 0.77 (+/- 0.03)
```

Fig 1.2 (KNN Euclidean) Worst Two Features



KNN (Extra Trees)

To demonstrate the difference in accuracy between best predictive features, and our worst predictive features we decided to train a KNN classifier only considering the best [Figure 1.1] and worst features [Figure 1.2] according to the scores Extra Trees. As we can see there is hardly any difference between the best and worst two features. Therefore, Extra Trees are giving the worst results.

```
Accuracy : (Distance = Euclidean | Neighbors = 1) : 0.66 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 3) : 0.69 (+/- 0.03)
Accuracy : (Distance = Euclidean | Neighbors = 5) : 0.71 (+/- 0.03)
Accuracy : (Distance = Euclidean | Neighbors = 7) : 0.73 (+/- 0.02)
Accuracy : (Distance = Euclidean | Neighbors = 10) : 0.75 (+/- 0.01)
Accuracy : (Distance = Euclidean | Neighbors = 12) : 0.75 (+/- 0.01)
Accuracy : (Distance = Euclidean | Neighbors = 14) : 0.75 (+/- 0.01)
```

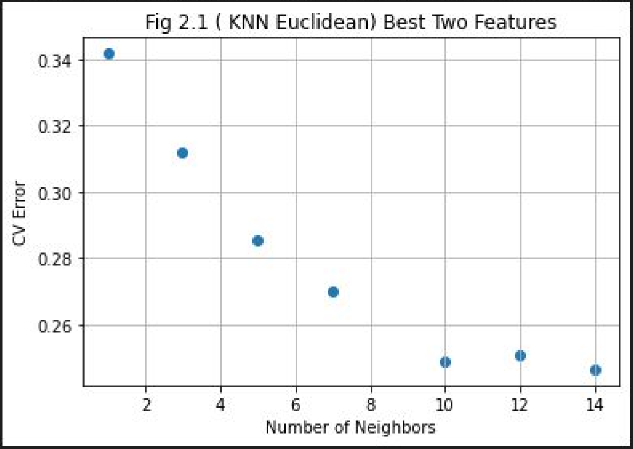


Fig 2.1 (KNN Euclidean) Best Two Features

```
Accuracy : (Distance = Euclidean | Neighbors = 1) : 0.60 (+/- 0.36)
Accuracy : (Distance = Euclidean | Neighbors = 3) : 0.75 (+/- 0.01)
Accuracy : (Distance = Euclidean | Neighbors = 5) : 0.75 (+/- 0.01)
Accuracy : (Distance = Euclidean | Neighbors = 7) : 0.76 (+/- 0.01)
Accuracy : (Distance = Euclidean | Neighbors = 10) : 0.76 (+/- 0.00)
Accuracy : (Distance = Euclidean | Neighbors = 12) : 0.76 (+/- 0.00)
Accuracy : (Distance = Euclidean | Neighbors = 14) : 0.76 (+/- 0.00)
```

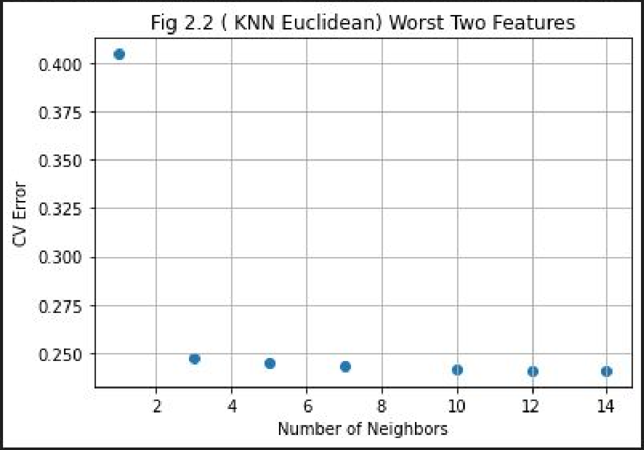


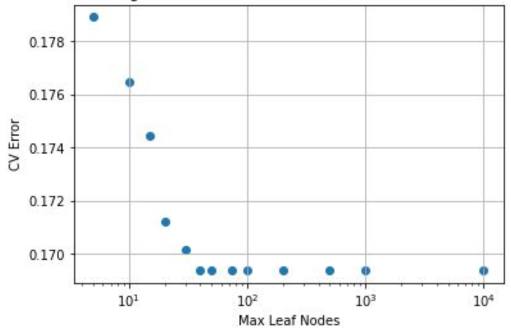
Fig 2.2 (KNN Euclidean) Worst Two Features

Decision Tree (K-Best)

The experiments were repeated with a Decision Tree classifier, again we see a stark difference between using our best features and our worst using K-Best.

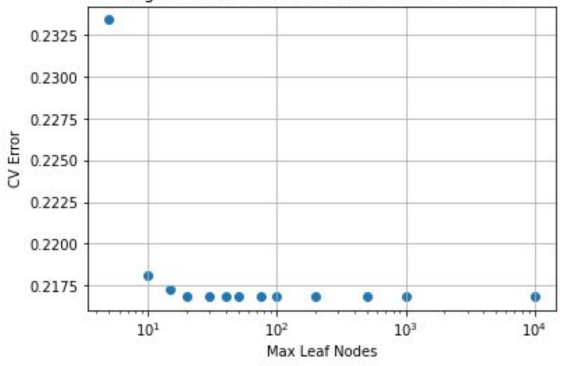
```
Accuracy : (Max Leaf Nodes = 5) : 0.82 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 10) : 0.82 (+/- 0.01)
Accuracy : (Max Leaf Nodes = 15) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 20) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 30) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 40) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 50) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 75) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 200) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 500) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000) : 0.83 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10000) : 0.83 (+/- 0.02)
```

Figure 3.1 (Decision Tree) Best Two Features



```
Accuracy : (Max Leaf Nodes = 5) : 0.77 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 15) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 20) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 30) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 40) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 50) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 75) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 100) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 200) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 500) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 1000) : 0.78 (+/- 0.02)
Accuracy : (Max Leaf Nodes = 10000) : 0.78 (+/- 0.02)
```

Figure 3.2 (Decision Tree) Worst Two Features



Decision Tree (K-Best)

The experiments were repeated with a Decision Tree classifier, there is hardly any difference between using our best features and our worst using Extra Trees. So far Extra Trees predicted the best two features inaccurately. Both best and worst features approximately have an accuracy of 76%.

Figure 4.1 (Decision Tree) Best Two Features

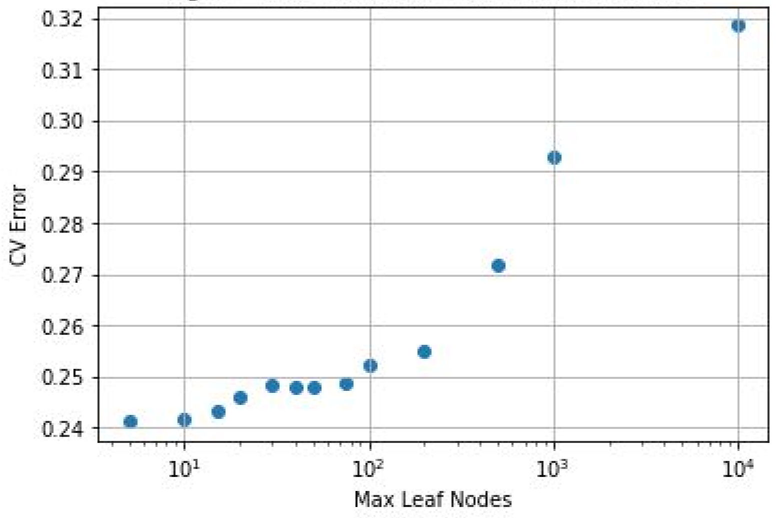
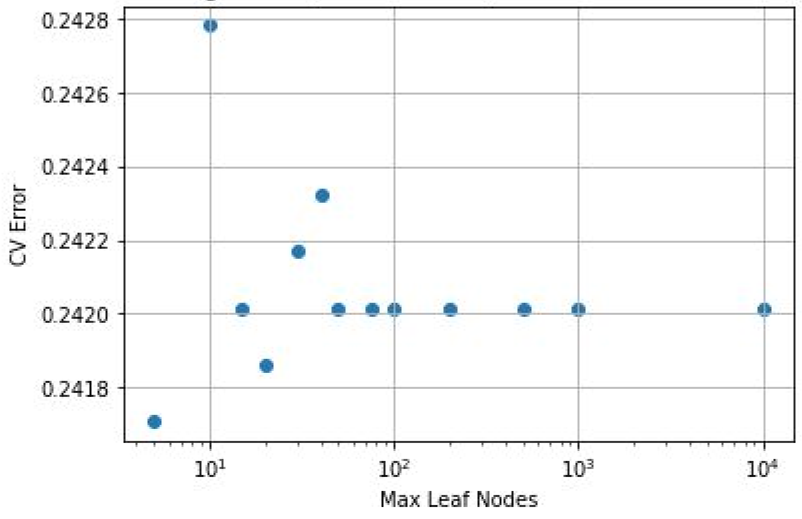


Figure 4.2 (Decision Tree) Worst Two Features



Gaussian Bayes(K-Best & Extra Trees)

Accuracy of Gaussian Naive Bayes according to Kbest two best features: 0.80 (+/- 0.01)
Accuracy of Gaussian Naive Bayes according to Kbest worst best features: 0.76 (+/- 0.02)
Accuracy of Gaussian Naive Bayes according to Extra Trees two best features: 0.76 (+/- 0.00)
Accuracy of Gaussian Naive Bayes according to Extra Trees worst best features: 0.76 (+/- 0.00)

Again the same results. K-Best did a better job determining the best two features while Extra Trees failed throughout all models used for accuracy. Based on the results we are going to conclude that the best two features according to K-Best are Capital Gain and Capital Loss.

Conclusion

Our conclusion is that there are certain features that are more predictive to an individual’s income. Our analysis says that Capital Gain and Capital Loss are the most predictive. Given that Capital Loss came in second we are suspicious about what the two factors actually say about an individual's income. The first thought could be that only those individuals that are willing to invest end up richer in the long run. But given that the threshold of this project is a mere $50k, and that the second more predictive feature is a net loss, we suspect that the model is actually saying that only people that Already had a salary of $50k are able to invest in non-liquid assets giving them weights in those most predictive features.

List of packages downloaded

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
from pylab import show
from sklearn.model_selection import cross_val_score


from sklearn.neighbors import KNeighborsClassifier # KNN
from sklearn.neural_network import MLPClassifier # Neural Network
from sklearn.tree import DecisionTreeClassifier # Decision Tree
from sklearn.ensemble import RandomForestClassifier # Random Forest
from sklearn.naive_bayes import GaussianNB # Gaussian Naive Bayes
from sklearn.decomposition import PCA, KernelPCA  # Kernel Principal Component Analysis

from sklearn.feature_selection import SelectKBest # Used for determining the strongest
relationship features with target
from sklearn.feature_selection import chi2 # Used for KBest non negative numbers
from sklearn.ensemble import ExtraTreesClassifier # Used for determining the strongest
relationship features with target
```

Citation for selection the best two features:
https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e