Lab 2: Sam Farren -> Knn Classifier on Iris and Income datasets

Section 1:

Lab 2 was a continuation of lab 1 in that in order to calculate the KNN classifiers and posterior probabilities, the similarity calculations from Lab 1 were used to find the K nearest neighbors to each point of the training data set. My approach was to take the original data set from lab 1 and use that as the test set, while taking the smaller data set given in lab 2 and use that as the training set. This was done for both of the iris and income data sets. This meant that when I computed similarities for each row of data, I took each test data instance(one of each of 150 instances in test data set), and compared it against every single row in the training data(all 70 in training data set) to obtain the k nearest neighbors. Once the K nearest neighbors were obtained for a row of data, I decided the prediction class by implementing a type of voting system. The class with the highest number of occurrences from the rows nearest neighbors was picked as the predicted class. I found this to be the easiest and most straight forward way to obtain the top k nearest neighbors to the testing point. The K nearest neighbors were then directly used to calculate the posterior probability. This was done by just taking the number of votes for each class, and then dividing by the k value. For example, if there were 7 votes for <=50K and 3 votes for >50K, the posterior probability with k=10 for my predicted class would be 70% for that specific data entry.

Success rates from both data sets were very different. I found it much easier to correctly predict the class for the Iris data set instead of the Income data set. I think this was due to the nature of the data, especially the income data. The income data required a lot of data manipulation, and transformation in order to obtain a similarity value. Examples of this include taking attributes such as age, or education level and putting them into bins rather than leaving the data alone. If you aren't particular about the bins you put each value into, then there is a chance for data loss. The same went for qualitative data, because it is hard to compare them to each other, so you need to transform it in order to obtain quantitative similarities. Qualitative transformations I performed I ended up doing were Education_cat, Marital_status, Race, Gender, Native_country, Capital Gain and loss, and hours per week. With a high number of transformations, this likely caused the decrease in performance on the income data set. This is where a lot of the problems came in because if you wanted a good performance measure, very precise transformations and decisions needed to take place. No transformations needed to be made on the Iris data set, so all of the data was preserved, and this is what resulted in a high success rate for class predictions.

Section 2:

A. Produce confusion matrices, classification and error rates for the test dataset for a few different values of k for both Iris and Income.

Iris:

k=1	Iris-Setosa	Iris-Versicolor	Iris-Virginica
Iris-Setosa	50	0	0
Iris-Versicolor	1	47	2
Iris-Virginica	0	8	42

k=5	Iris-Setosa	Iris-Versicolor	Iris-Virginica
Iris-Setosa	50	0	0
Iris-Versicolor	0	48	2
Iris-Virginica	0	8	42

k=15	Iris-Setosa	Iris-Versicolor	Iris-Virginica
Iris-Setosa	50	0	0
Iris-Versicolor	0	50	0
Iris-Virginica	0	13	37

The following confusion matrices were produced for the iris data set with k equal to 1, 5, and 15. For all three instances, the accuracy of Iris-Setosa was 100%. Iris-Versicolor had a very high success for prediction also, but almost all the incorrect predictions came from classes that were Iris-Virginica. In particular, every wrong instance of Iris-Virginica was predicted to be Iris-Versicolor. All 13 instances for when k=15 were falsely classified as Iris-Versicolor, which may have been an issue caused by both classes having similar data that was difficult for the k nearest neighbors to differentiate.

Income:

k=5	Predicted: <=50K	Predicted: >50K	
Actually: <=50K	190	230	420
Actually: >50K	17	83	100
	207	313	

k=25	Predicted: <=50K(NO)	Predicted: >50K(YES)	
Actually: <=50K(NO)	300	120	420
Actually: >50K(YES)	35	65	100
	335	185	

k=50	Predicted: <=50K(NO)	Predicted: >50K(YES)	
Actually: <=50K	324	96	420
Actually: >50K	55	45	100
	379	141	

These confusion matrices were produced for the income dataset with k equal to 5, 25, and 50. Based off these choices of k it is clear to see that with a low k value, there is a high number of false negatives where I predicted the class was >50K but it was actually <=50K. As K increased, it is notable to mention the increased accuracy for correctly predicting rows that had a class of <=50K. However, since data in the test data set was skewed approximately 81% for entries that had class <=50K and 19% for entries that had class >50K. This could have just resulted in an increased prediction of one particular class and not the other.

B.

All Calculations based off of k = 25

- True Positive (sensitivity): TP/P = TP/(TP+FN)= (300+65)/(300+65+120) = 75.26%
- False Positive: 1-specificity = 1 - TN/(FP+TN) = 1 - (324)/(324+96) = 22.9%
- True Negative(Specificity): TN/N = TN/(FP+TN) = (324)/(324+96) = 77.1%
- False Negative: 1-TPR = 1-TP/(TP+FN) = 1- (300+65)/(300+65+120)
- Recall: TP/(TP+FN) = (300+65)/(300+65+120) = 75.26%

- Precision(PPV): TP/(TP+FP) = (300+65)/(300+65+96) = 79.18%
- F-Measure: 2TP/(2TP + FP + FN) = (2*365)/(2*365 + 96 + 120) = 77.17%
- **ROC Curve: Couldn't get it working with Scikit, it was giving me many problems**

C. Provide detailed analysis of the results. What trends did you observe? Provide graphs and/or statistics to back up and support your observations.

Some of the trends observed from the results of this lab include the high predictability of true negatives within the data. This however, I think was due to the fact that 81% of the data had a class <=50K (Negative) and therefore if most of the data was classified as Negative then the true negative rate would indeed be higher. Another thing worth mentioning about the iris dataset, is that it was very easy to predict the classes that were Iris-Setosa. Based off of statistical analysis, this was due to the particular class, since its' values tended to be in the lower third of data. Therefore, there weren't many neighbors that were misclassified near it. If I had more time I would have shown a contour plot of it demonstrating the neighbors of different k values, and highlighting the iris setosa classes standing out from the rest.

D. You implemented 2 different proximities in homework #1. If you change the proximity, do the prediction results change significantly?

The two proximity calculations I chose to implement in lab 1 were a Minkowski and cosine approximation for the Iris data set. I decided to switch the Minkowski proximity to a more specific version of it and use a euclidian distance instead. The euclidian distance proved to be a very good predictor for classifying unknown data. For a cosine proximity however, the success of prediction was much much lower. The following comparisons were made for the Iris data set:

	Euclidian Distance Success	Cosine Succcess
k=5	93.33%	39.33
k=7	95.33	62.66
k=9	94.0%	63.33
k=15	91.33	53.33

From the table above, it is very easy to tell that the euclidian distance proximity did a much better job of predicting classes. The cosine proximity maxed out around 63.33% when the k value was equal to 9. It seemed to have trouble predicting the Iris-virginica class, which was seen earlier in the confusion matrices for the euclidian distance implementation. For the income dataset, I didn't get the cosine implementation working all the way. However, based off the drastic change in performance on the iris data set, one could expect to see a high decrease in performance on the income data set as well for a cosine implementation.

E. When you vary the number of nearest neighbors k, how does the performance on test data change? What value of k do you recommend for these datasets?

Iris:
I tested to see what values of k provided the highest accuracy prediction. The following table was obtained from the Iris test data set.

k	Accuracy
1	92.67%
2	94.0%
3	93.33%
4	96.0%
5	93.33%
6	94.67%
7	95.33%
8	96.0%
9	94.0%
10	95.33%
15	91.33%
20	90.67%

While 90% success is considered good for predicting the classes or unseen data, anything less than 20 and greater than or equal to 3 satisfies this. If you want to maximize the performance, then a k value between 7 and 10 (inclusive) will maximize the performance. These k values will reduce the sensitivity to noise points, but also won't be big enough to include other data points as nearest neighbors.

Income:

k	Accuracy
1	24.81%
5	52.5%%
10	37.5%
15	52.31%
20	62.5%

k	Accuracy
25	70.19%
31	69.81%
39	70.0%

For the income data set, the ability to predict was much much lower than for the iris data set. In particular, low values of k produced very unreliable results, stemming from the amount of noise produced from other points. Therefore, to obtain a higher success rate, k had to be increased to above 20 to get somewhat consistent results. Based off the table above, I would say a k value between 25 and 30 will give the most consistent and maximize success of predicting classes. Any lower produces too much noise, and any higher doesn't work because of the uneven split in data, with the class >50K only holding 20% of the total rows in the data set.

F. Did you need to make any changes from Homework #1 because your algorithm was not working well on the test data?

Yes, I needed to make a couple of changes for the iris data set. I originally implemented a Minkowski approximation for the similarity function, but decided to switch to a Euclidian Distance approximation. The code was changed to the following:

```
    euclidian = pow(pow((firstRow[0] - row[0]), 2),.25) +
pow(pow((firstRow[1] - row[1]), 2),.25) + pow(pow((firstRow[2] -
row[2]), 2),.25) + pow(pow((firstRow[3] - row[3]), 2), .25)
```

This made a significant impact on the accuracy obtained increasing from 82.61% to 93.33% for k=5, as well as k=1 and k=15.

For the income dataset, it was a much more difficult problem to solve because again, the data was much more complex and required a lot of transformations. I decided to just keep my similarity calculation and use it as is in computing the K nearest neighbors, even though after analyzing the results the success rate of prediction was not optimal (maxing out around 70%).

Section 3

Custom Python Comparison with R:

I decided to compare my own python implementation with one in which R provides via its API. I first tested against the Iris dataset to see how it would match up, since mine did pretty well being able to predict 96% of classes correctly with k=4 and k=8. The following summary was produced in R for the Iris data:

	iris_pred			
iris.testLabels	Iris-setosa	Iris-versicolor	Iris-virginica	Row Total
Iris-setosa	12	0	0	12
	1.000	0.000	0.000	0.300
	1.000	0.000	0.000	
	0.300	0.000	0.000	
Iris-versicolor	0	12	0	12
	0.000	1.000	0.000	0.300
	0.000	0.923	0.000	
	0.000	0.300	0.000	
Iris-virginica	0	1	15	16
	0.000	0.062	0.938	0.400
	0.000	0.077	1.000	
	0.000	0.025	0.375	
Column Total	12	13	15	40
	0.300	0.325	0.375	

```
> iris.training <- iris[ind==1, 1:4] iris.test <- iris[ind==2, 1:4] - Read
```

The results for this implementation were very good, producing only one wrong prediction of the 40 in the training set. The only one that was wrong was an Iris-Virginica predicted as an Iris-Versicolor.

For the income dataset, it was a little more difficult even to use an off the shelf implementation because the data transformations still had to occur. I also did some research to try and find the best way to approach this because I felt that my implementation may have used some unnecessary attributes for finding the nearest neighbors to predict the class. It was then decided to only keep Capital Gain, Marital Status, Education_cat, Age, Occupation, and Income Level for the calculation. After splitting and training the data set, the results were produced for k=5,7, and 9. The following table was produced:

.

k	Accuracy
5	0.8307116
7	0.8343742
9	0.8360932

It is easy to see that compared with my implementation in python, this is much better. The accuracy approached about 84%, where mine with the same k value was around 40%, not even half of the R implementation. Even if I changed my model to only include the six classes listed for the R implementation, I don't think mine would have even come close in relation to success rate. In summary, the R implementation was a better predictor for class variables on the income and iris dataset.