HOMEWORK 1

Class: CS5785

Authors: Anna Guidi, Samantha Yip

September 13th, 2017

A. Purpose of the Study

Problem 1 - Digit Recognizer: Learn how to program, implement and measure the accuracy of a K-NN classifier using the Digit Recognition dataset found on Kaggle.

Problem 2 - Titanic: Clean, format and manipulate the data of the *Titanic: Machine Learning From Disaster* dataset, and subsequently make a Logistic Regression model to predict whether or not the passengers may have survived the shipwreck.

B. Procedure

Problem 1 - Digit Recognizer

1. After importing the MNIST data set, we parse through the data and make an *N x p* array for the pixel values of each number-image with the *.iloc* method (through the pandas library), where *N* is the number of samples, and *p* is the number of attributes per sample. We also create a label vector *y* containing the actual label (digit) of the sample using the same library and method. This is the data we will be working with.

2. Next, we want to see what each digit looks like. The first occurring sample of every MNIST digit is identified using the np.unique method, put in an array, and subsequently reshaped into a 28X28 matrix. Each digit in the array is then displayed using the matplotlib library, as can be seen in figure 2.

3. We then examine the *prior probability* of each class by looking at the distribution of all digits in the data set. This is done by means of a normalized histogram, which is visible in figure 1.

4. Before implementing K-Nearest Neighbor, a good exercise is to find the best match for one digit of each kind simply by finding its closest match in the training data based on Euclidean distance. In order to do that, for each sample digit in the array mentioned earlier, we iterate through the entire data set with a *for loop* and compute the distance between it and all other data points, storing a *tuple* consisting of the *distance and the index (or row)* number in a new array of distances. These are then appended to another array which eventually contain all possible distance-row number tuples for each digit. We iterate over this last array and retrieve the tuple with the minimum distance value for each sample digit and print its distance and row number. In the final step, we iterate over the array of closest matches and array of sample digits simultaneously, and check if their labels actually coincide, otherwise adding an asterisk next to the label of the closest neighbor.

5. In order to visualize some data more concretely, we first identify and then iterate through all digits labelled as "0" and compute the pairwise distances to other 0s. The same procedure is done for all digits labeled "1". The "imposter distances" are also computed, meaning the distances between all possible pairs of 0 and 1. Rather than using a for loop which would take a long time, it is more efficient to use the *metrics.pairwise.euclidean_distances* function which performs matrix operations. The histograms of these three distances are plotted first individually, and then together on the same set of axes (figure 3), with the genuine distances for 0 and the genuine distances for 1 combined into one array.

6. We make an ROC curve based on the above-mentioned set of distances in order to visualize and compare the true positive rate versus false positive rate for different possible cutpoints, which should span from the lowest distance recorded (found in the genuine distances array) to the highest distance (most likely found in the imposter distances array). For the sake of speed, we only use a subset of the data to compare each genuine and imposter value to every value of tau, and classify each point as a true positive, a false negative, a false positive, or true negative. The accumulation of these counts is then used to calculate the necessary rates. The Equal Error Rate and random error rate are calculated and discussed in the results section. The ROC curve can be seen in figure 4.

7. We now construct a K-NN classifier by making a function that takes in the data point of the digit we would like to classify (let us call it x), the array of data we would like to compare this point to, the labels of all the said data, and a value for k, in this case 3. We iterate with a for loop through all of the already-classified data points and compute the Euclidean distance to the point x. The data points are then sorted from lowest to highest value based on their distance to x, and ultimately the k (3) data points with the lowest distances to x are retrieved. The final step is to find the *mode of the labels* of these k data points and set this equal to the predicted label of x.

8. Now that we have created a K-NN classifier, we divide our training data into 3-folds in order to do cross validation and feed the testing data of each fold into the K-NN function we constructed in the earlier step. In addition to the predicted labels that are returned by the function, we also have the set of actual labels for each point. This means that it is possible to compare the actual label versus the predicted label for each point in a fold, do a count of accurate predictions and false predictions, and then calculate the accuracy ratio of the K-NN classifier for each fold. The overall accuracy across all 3 folds is described in the results section.

9. Another way to understand how well the K-NN works other than looking at the accuracy percentage, is to display it visually by means of a confusion matrix. This is relatively simple to

implement, as we run a set of data points whose labels are known to us through the classifier function in order to retrieve a complimentary set of predicted labels. Both the actual and predicted labels are then used to construct the confusion matrix (figure 5).

10. We run a subset of the testing data from Kaggle on our K-NN classifier.

Problem 2: The Titanic Disaster

1. We import the following libraries: *numpy, pandas, seaborn, matplotlib* and *sklearn*.

2. We use the *read_csv* function in the *pandas* library to put the training and testing data for titanic in a data structure, and name the columns of the training and testing data.

3. We validate the data through the csv file, and see that the *'Age', 'Cabin'* and *'Embarked'* columns have empty values. There are some empty values for certain features: In the training data, '*Age'* has 177 empty cells, '*Cabin*' has 687 empty cells, and *'Embarked'* has 2 empty cells. In the testing data, '*Cabin*' has 327 empty cells, and *'Fare'* has 1 empty cell.

4. Since we thought that *'Age'* is an important feature for predicting the survival of passengers, we want to fill in the empty cells for the logistics regression. We cannot delete the empty cells, because 177, which is the number of 'Age' empty cells, is a big number. We plot a box plot, with the x-axis as *'PClass'* and the y-axis as *'Age'*, and notice a pattern between 'Age' and *'PClass'*. Then, we obtain the mean of age within each '*PClass'*. Based on the passenger's *'PClass'*, we estimate the empty cells under the *'Age'* column using the mean age for the associated *'PClass'*. To fill in the *'Age'* empty cells, we create a function called *age_approx* that take in *'Age'* column as an argument, and applied the function to the entire *'Age'* column.

5. We use the drop function to remove the *'PassengerId', 'Name', 'Ticket'* and *'Cabin'* columns, which, we believe, are unnecessary for the logistic regression, because almost each passenger has a unique *'PassengerId', 'Name', 'Ticket'* and *'Cabin'.*

6. We use the dropna function to remove a small number of empty cells remaining in the dataset, which include 2 empty cells under the training data's *'Embarked'* column and 1 empty cell in the testing data's *'Fare'* column.

7. We double check that the entire dataset does not have empty cells by using the isnull() and sum() functions.

8. We use the iloc[].values to put the both training and testing data into lists and convert the lists into arrays called *training_data* and *testing_data* (the purpose of conversion is to use numpy functions for better analysis).

9. We use the labelencoder_X.fit_transform function to convert any string values into an integer. For example, 'male' and 'female' turn into binary numbers, which in our case, 'male' is 1 and 'female' is 0. Also, the string values under 'Embarked' column turn into integers.

10. We make all the values within the *training_data* and *testing_data* arrays into floats, so that we can fit the arrays into the logistic regression function.

11. We separate the labels within the training data into the *train_y* list and the features into the *train_x* list. With the train_test_split() function that has a test_size of 0.7, we do a cross-validation by putting 0.7 of the dataset into the testing bucket and 0.3 into the training bucket, since we do not have labels within the training data. In our case, X_train contains the features of the training bucket obtained from the training data and Y_train contains the labels of the training bucket obtained from the training data. X_test contains the features of the testing bucket obtained from the training data and Y_test contains the labels of the testing bucket obtained from the training data.

12. With LogisticRegression() and fit() functions, we create a prediction model called LogReg based on the X_train and Y_train. Then, we use LogReg.predict() to predict whether the passenger within X_test would survive the disaster, and call the result list y_pred. To check the accuracy score for our model, we use the sklearn.metrics.accuracy_score() function on Y_test and y_pred.

13. If the accuracy score is decently high, we use the LogReg.predict() on the *testing_data* to predict whether the passengers within the testing data would survive the disaster.

C. Results

Problem 1: Digit Recognizer

3. Prior Probability of all digits

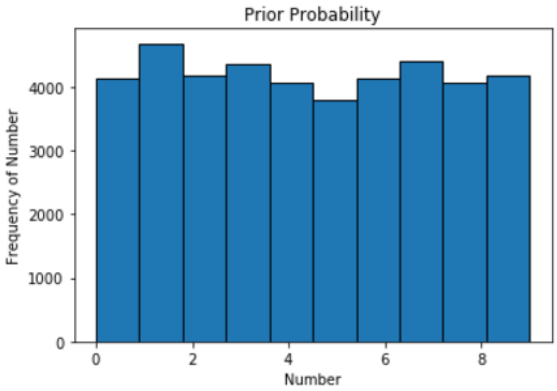

Figure 1: Prior Probability Distribution

Is [the distribution] uniform across the digits?

The distribution is not uniform, we have a higher number of 1s, 3s and 7s, and a lower number of 5s. The distributions for each digit are similar enough where it is reasonable to assume that this will not affect the closest match or K-NN algorithm too strongly, although with a more extreme discrepancy in frequency, this could potentially impact what qualifies as a best match.

4. Find the closest match (based solely on Euclidean distance) for one example of each digit

All sample digits had a closest match with a data point that has the same label with the exception of 3, whose closest neighbor was 5, which was an unexpected result.

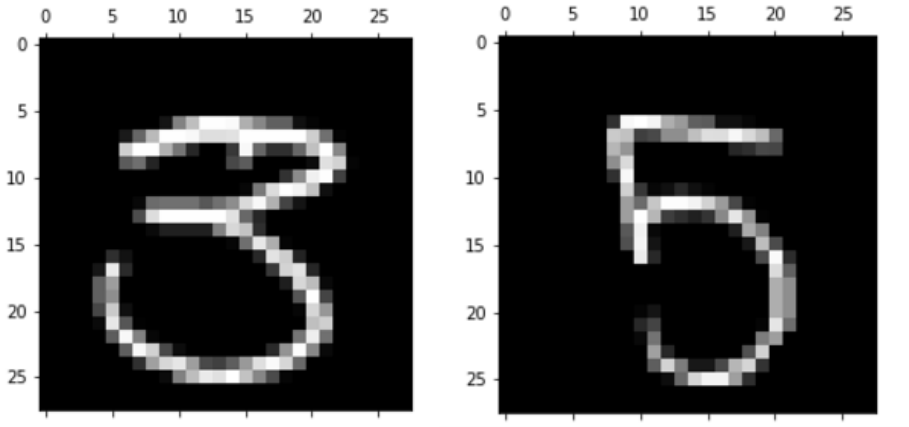

Figure 2: displays of MNIST digits

All sample digits had a closest match with a data point that has the same label with the exception of 3, whose closest neighbor was 5, which was an unexpected result. Another very interesting result to keep in mind from this exercise is that the digit 2 had an extremely close match to another 2 with a distance value of 489.67, whereas other than the digit 8 (which has a closest match with a distance of 863.5), all other close matches where over 1000. While it is not possible to come to any statistically significant conclusion from this, it is something to keep in mind.

5. Histogram of genuine vs. Imposter distances

Figure 3: histogram of Imposter versus Genuine distances

Both the imposter distances (in orange) and the combined genuine distances (in blue) have a tendency towards normal distribution, although it is important to note that the imposter distances have a much tighter distribution spanning from ~2000 to ~4000, with much higher frequencies for values that span 2500 and higher. The genuine distances unsurprisingly start at a much lower value, from ~400 to ~3500, but there is a lot more variation and additionally a significant amount of overlap between both arrays between the distances of ~2000 and ~3500, which might make it difficult to come up with a good cutoff point.

6. ROC curve

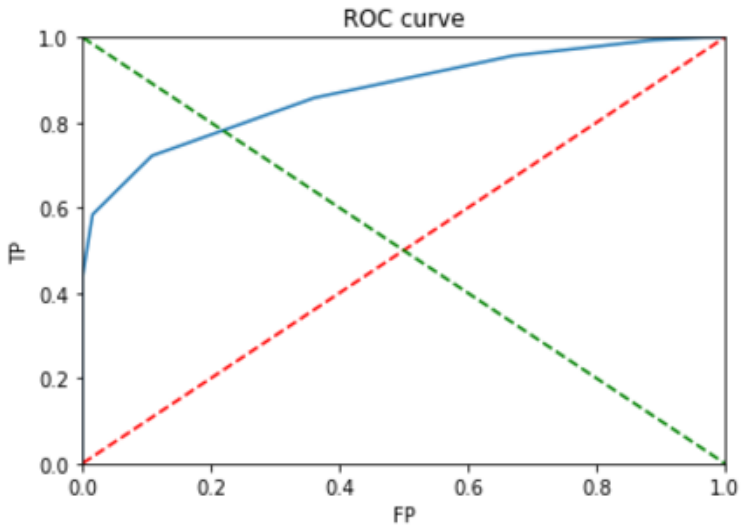

Figure 4: ROC curve

As can be seen from the ROC curve, for smaller values of tau, otherwise known as the cutoff point or threshold, the false positive rate is much lower. There is a trade-off between FP and

TP. A "good" ROC curve would hug the y axis and be significantly above the red dotted line, which would be the ROC "curve" of a random guess. Our ROC curve is as expected.

What is the equal error rate?

The Equal Error Rate (ERR) is the value where the false acceptance rate is equal to the false rejection rate. The lower the ERR is, the higher the accuracy of the prediction model. Our because our data is not granular enough, our EER value is somewhere between 0.27795 and 0.10803, so we estimate that it could be ~**0.19299.**

What is the error rate of a classifier that simply guesses randomly?

The error rate of a classifier that guesses randomly is 50%, since there is an equal chance of a genuine distance or imposter distance being classified correctly or incorrectly.

7. KNN classifier

When we run the first 400 data points of the testing set through our KNN classifier, we get an accuracy of .8, which improves when we test for more points (see cross validation below).

8. Cross validation

Report your average accuracy

The average accuracy of the cross validation depends on how many data points we choose to run. With a very low data set (100 points), we only get an accuracy of around 54%, but even increasing that to 400 data points we improve our accuracy dramatically to around 88%. With **21,000** data points we obtain an accuracy of **95.88%**. While it is important to have a healthy sample size, the increase in accuracy tapers off after a critical number of data points.

9. Confusion Matrix

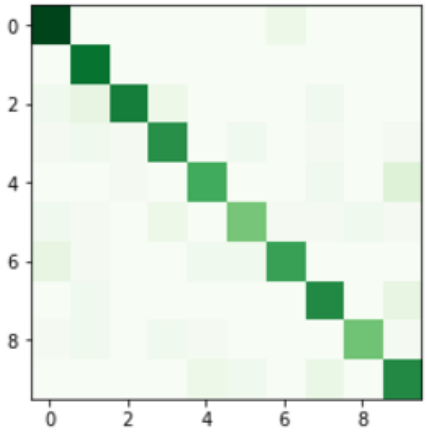

Figure 5: Confusion Matrix

Which digits are particularly tricky to classify?

The above confusion matrix (based on a small subset of a couple hundred data points) functions as a sort of heat map, with the deeper shades of green corresponding to higher number of matches, which makes sense because if we have the darkest shades of green across the diagonal. This is a positive sign because it means K-NN is labelling the data correctly (a 2 as a 2 and so forth). It seems from this matrix that digits 1 through 3 inclusive had the highest number of correct matches, whereas the digit 5 seems the trickiest to classify, and the digit 4 sometimes gets misclassified as a 9.

It is worth noting however, that when K-NN is run with a bigger data set (we ran it with a few thousand data points), the diagonal squares intensify with an even deeper green and the rest of the squares become pale (white), meaning that the accuracy generally increases with a larger dataset (as further discussed in the conclusion).

Problem 2: Titantic Disaster

1. Box Plot

The box plot, figure 6 as shown below, is used to estimate the age of passengers who have missing information under the 'Age' column. In the box plot, the range of age is small and concentrated within each Pclass, and we decided Pclass is a good estimator for the age. Based on plot, it tells us that the average age for Pclass = 1 is 37, 28 for Pclass = 2 and 24 for Pclass = 3.

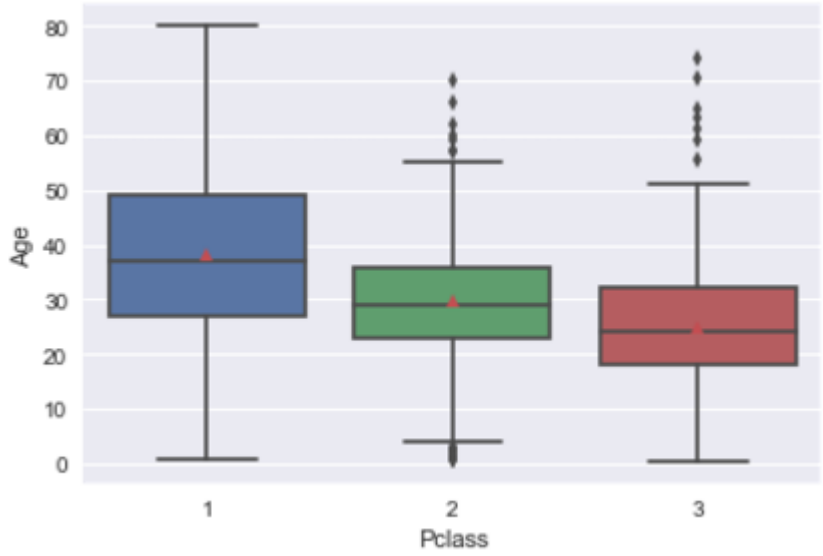

Figure 6: Boxplot for PClass vs Age

2. Logistic Regression

Our prediction rate for the testing data using Logistic Regression is 79%, which is a high accuracy rate. Thus, it is appropriate to use our model based on the features: PClass, Sex, Age, SibSp, Parch, Fare, Embarked. These features are reasonable, because the sex and age matter within the Titanic disaster. Women and children have a higher chance of being saved in the

situation. Moreover, Fare and PClass also matter, since the higher social status you are, the higher chance you are being saved. Lastly Parch, SibSp and Embarked do have an effect on predicting the outcome of the passenger. For example, if you have a Parch = 2, it means that you might be a child with two parents on the ship. Embarked might tell us the social status of a passenger, since further destinations would require a more expensive ticket, and passengers who can afford the ticket might be in a higher social status. We decided to ignore Cabin, PassengerId, Ticket and Name, because each of these feature might be unique for each passenger, and it does not give us information for predicting the survival rate of a passenger. In our final prediction on the testing data, 288 people survived out of 417 passengers, which is around 70% of the population in testing data.

D. Conclusion

How well your solution works

Our solution works well when we look at the output accuracy, but we were unable to run all of the testing and training data for K-Nearest Neighbor and had to settle for running only a subset.

When running large amounts of data through the KNN classifier, *for* loops are not very efficient to iterate through large datasets, and slow down the process significantly. It is therefore useful to avoid for loops when possible and know which methods are out there to make computations faster (such as the metrics.pairwise.euclidean_distances method). It would also be helpful to set up parallel processing in the future or use AWS to compute more data.

Regarding the logistic regression for the Titanic dataset, our prediction rate for the testing data is about 79%. The accuracy rate is high, which shows that our model is appropriate for application on the testing data. Since all of our features are backed up by reasons on how it will affect the outcome of the passenger's survival, we believe our model works well and is reliable.

Any insights you found

KNN - Digit Recognizer

Some insights are that the larger the dataset, the slower the processing time, so it is worth knowing which libraries and tools out there to make calculations much faster. If we were to redo this lab, we would keep in mind efficiency of dealing with large amounts of data. We would also be careful in the future to perhaps pick a better subset than the first X number of rows, and instead pick random, but even number of samples for each digit.

In addition, while it is true that the bigger the data set is, the more accurate the data and better your predictions, but predictably the net accuracy gain tapers off very quickly. As discussed before, some digits are more unique and thus easier to identify than others.

Logistic Regression - Titanic

There are creative ways to make up for missing data in logistic regression by making comparisons between the missing data points of a certain category and other, more complete data, and picking a value that is more refined than a net mean. We also learned that logistic regression is a good model for predicting the binary outcome of testing data, if we reasonably select the appropriate features as the input of the model.

F. Citations

- https://www.udemy.com/machinelearning/learn/v4/t/lecture/5683430?start=0
  Note: the above video is part of an online introductory ML course and is probably only visible to those who pay the fee to access the course. I referenced some initial code that was contained in the video Part 1: Data Preprocessing – Importing the Dataset, for the purpose of learning how to use the Encoder for the Titanic problem (Problem 2).

- http://www.data-mania.com/blog/logistic-regression-example-in-python/
  The above link was used in particular to help estimate the missing "Age" fields for the Titanic dataset in Problem 2.