ASSIGNMENT 1

September 18, 2019

Bar Kadosh (bk497), Ben Kadosh (bk499)
CS5785: Applied Machine Learning
Instructor: Nathan Kallus, Teaching Assistants: Xiaojie Mao, Yichun Hu

Contents

0.1 Summary

Our general approach to the coding sections of this assignment was to first organize the data into an easy to use and readable format. To do so, we utilized the pandas and numpy libraries. Furthermore, we utilized numerous functions from the sklearn library to analyze the data and answer questions.

For the Digit Recognizer exercise, the main aspects of focus were how to efficiently calculate the distances between the images, better understanding how distances were influencing genuine/imposter matches in parts e/f, and how to effectively find the k nearest neighbors for any given k and validate the results. Ultimately, we produced positive results: our KNN classifier correctly labeled 96.8 percent of the test points.

For the Titanic exercise, the main aspects of focus were understanding all of the available variables, cleansing and transforming the data to be more standardized and provide greater insights, and to reason through which variables were more/less important to the regression. The final steps for Titanic were to test different combinations of variables on the training data to gain comfort with the variables we chose and then refining those variables on the test data to make sure we didn't overfit our model to the training data. Again, our results were positive: our logistic regression classifier correctly labeled 77.99 percent of the test points.

0.2 Problem 1: Digit Recognizer

(a) Join the Digit Recognizer competition on Kaggle. Download the training and test data. The competition page describes how these files are formatted.

(b) Write a function to display an MNIST digit. Display one of each digit.

Our function takes in a dataframe and numerical value as arguments, filters the data frame for the value passed, and plots the image for the first row that corresponds to that value. We then plot the 10 subplots (1 subplot for each possible label) on a figure, which is attached below.

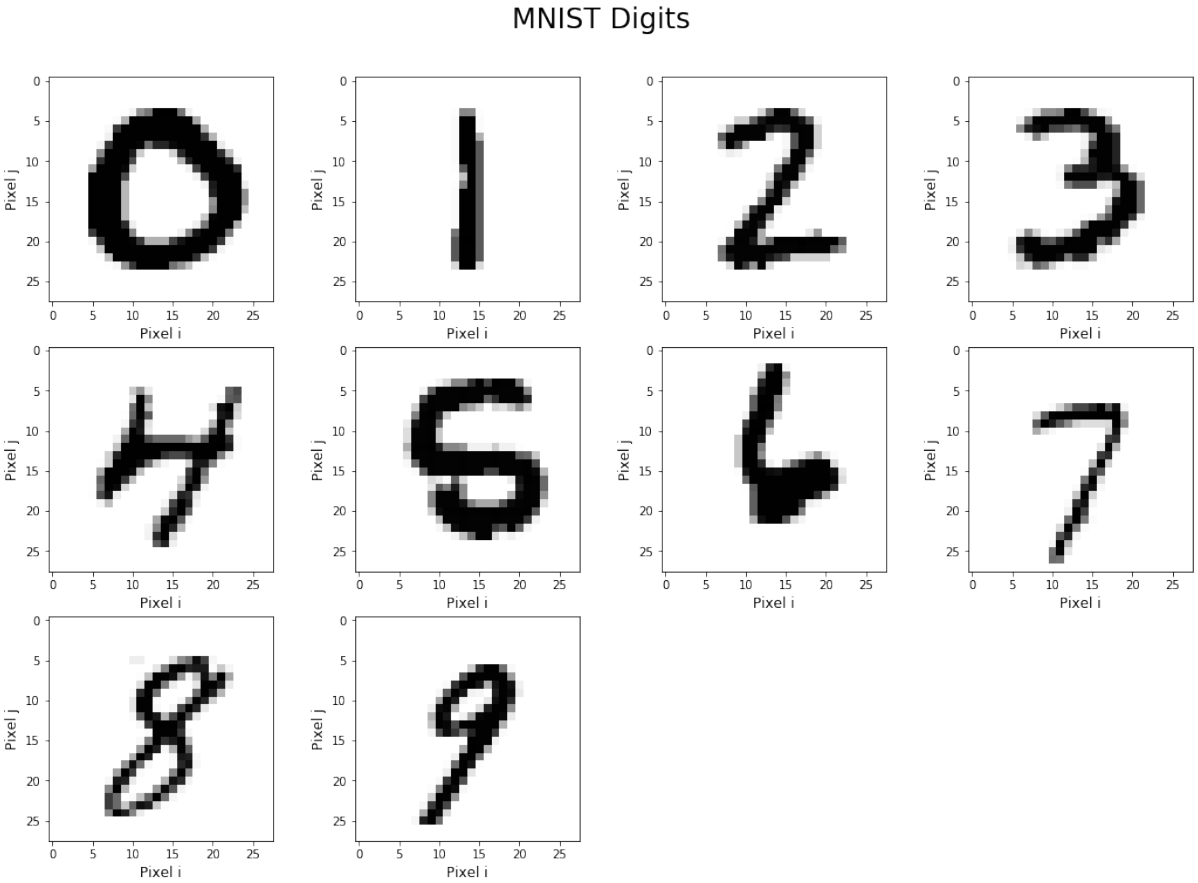

Figure 1: MNIST Digits Plot

(c) Examine the prior probability of the classes in the training data. Is it uniform across the digits? Display a normalized histogram of digit counts. Is it even?

The normalized histogram of the digit label counts for each of the digits 0 through 9 is roughly even at about 10 percent each – fluctuation from 9 to 11 percent can be seen for some values. However, generally speaking, the distribution is fairly uniform as most values' normalized counts hover around 10 percent.

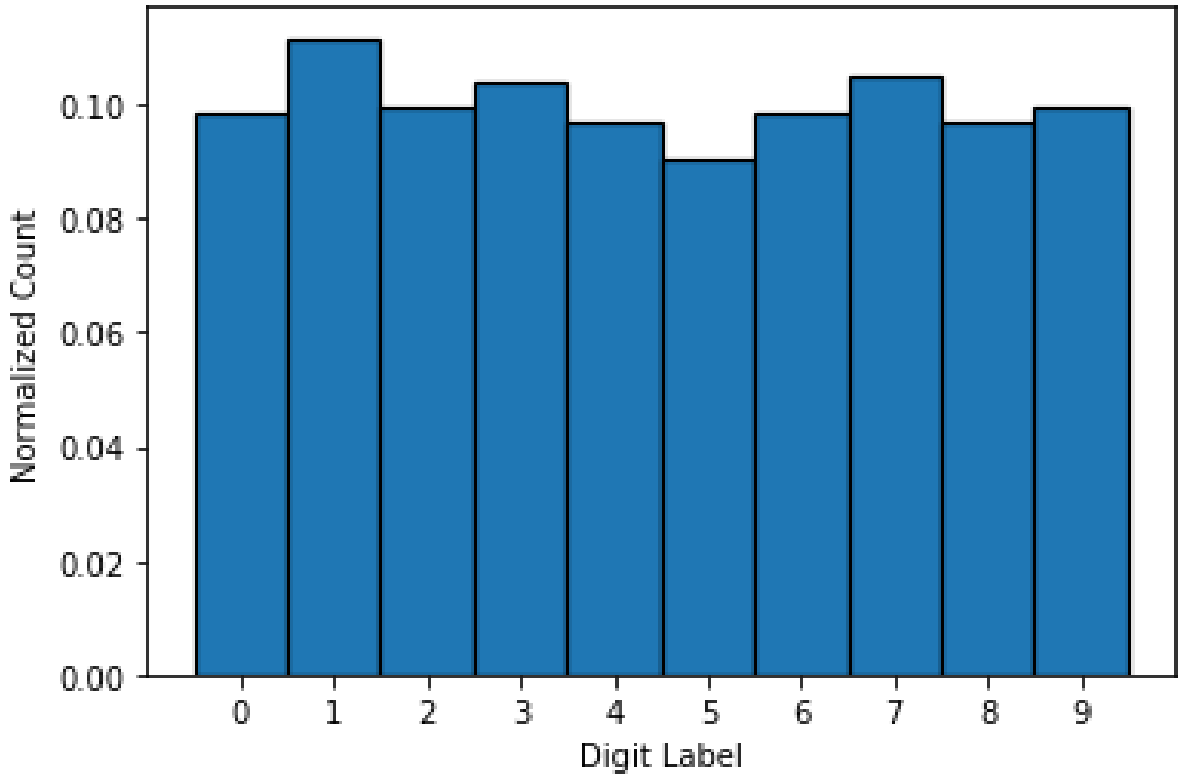

Figure 2: Normalized Digit Count Histogram

(d) Pick one example of each digit from your training data. Then, for each sample digit, compute and show the best match (nearest neighbor) between your chosen sample and the rest of the training data. Use L2 distance between the two images' pixel values as the metric. This probably won't be perfect, so add an asterisk next to the erroneous examples (if any).

For this problem, we defined a distance function that takes in a dataframe and a row. For the input row, the distance is calculated to all other rows in the dataframe and the index of the minimum distance is found (ignoring distance to itself). The function then returns an array of the index of the closest neighbor and the distance to that neighbor.

Then, we created a for loop that found the first row that contained a label of each digit. For each row (each with a unique digit label), we called our "dist" function to find the index of the row closest to the row in question. Finally we printed the actual and closest neighbor labels, their distance from each other, and put asterisks around the row if the labels didn't match.

```
     Actual Label:  0 ,  Closest Neighbor Label:  0   Distance:  1046.5954328201515
     Actual Label:  1 ,  Closest Neighbor Label:  1   Distance:  489.67948701165744
     Actual Label:  2 ,  Closest Neighbor Label:  2   Distance:  1380.877257398354
***  Actual Label:  3 ,  Closest Neighbor Label:  5   Distance:  1832.6649993929605 ***
     Actual Label:  4 ,  Closest Neighbor Label:  4   Distance:  1356.8809822530493
     Actual Label:  5 ,  Closest Neighbor Label:  5   Distance:  1066.3676664265472
     Actual Label:  6 ,  Closest Neighbor Label:  6   Distance:  1446.5113203843239
     Actual Label:  7 ,  Closest Neighbor Label:  7   Distance:  863.5010133172977
     Actual Label:  8 ,  Closest Neighbor Label:  8   Distance:  1593.7775879965184
     Actual Label:  9 ,  Closest Neighbor Label:  9   Distance:  910.5767403135224
```

Figure 3: Nearest Neighbor Distances Function

As can be seen in this example, the closest label for one of the rows with label 3 was 5, which is not the correct label. However the rest of the closest neighbors did match up with their corresponding test sample.

(e) Consider the case of binary comparison between the digits 0 and 1. Ignoring all the other digits, compute the pairwise distances for all genuine matches and all impostor matches, again using the L2 norm. Plot histograms of the genuine and impostor distances on the same set of axes.

For this problem, we started by calculating a distance matrix for each point to all other possible points in the 0-1 dataframe (with all pixels). We did a similar computation for our array of labels – the result was a matrix such that a value in the matrix with a 0 meant the pair was a genuine match and a value in the matrix with a 1 meant the pair was an imposter match.

Using these corresponding matrices, we then filled a list with all of the genuine points (by referencing the indices in the label matrix with value 0) and another list with all of the imposters (by referencing the indices in the label matrix with value 1). Finally, we plotted a histogram with the distances for genuine matches and the distances for imposter matches.

The histogram shows the genuine matches in a light blue color and the imposter matches in light orange. The overlaps between genuine and imposter matches are highlighted in a light gray, showing that there isn't a defined split between genuine and imposter matches by distance. However, the histogram does show that the center of the genuine distribution falls around a distance of 2000, while the center of the imposter distribution falls very close to a distance of 3000. This highlights that a difference in distance values does statistically exist between many imposter and genuine matches.

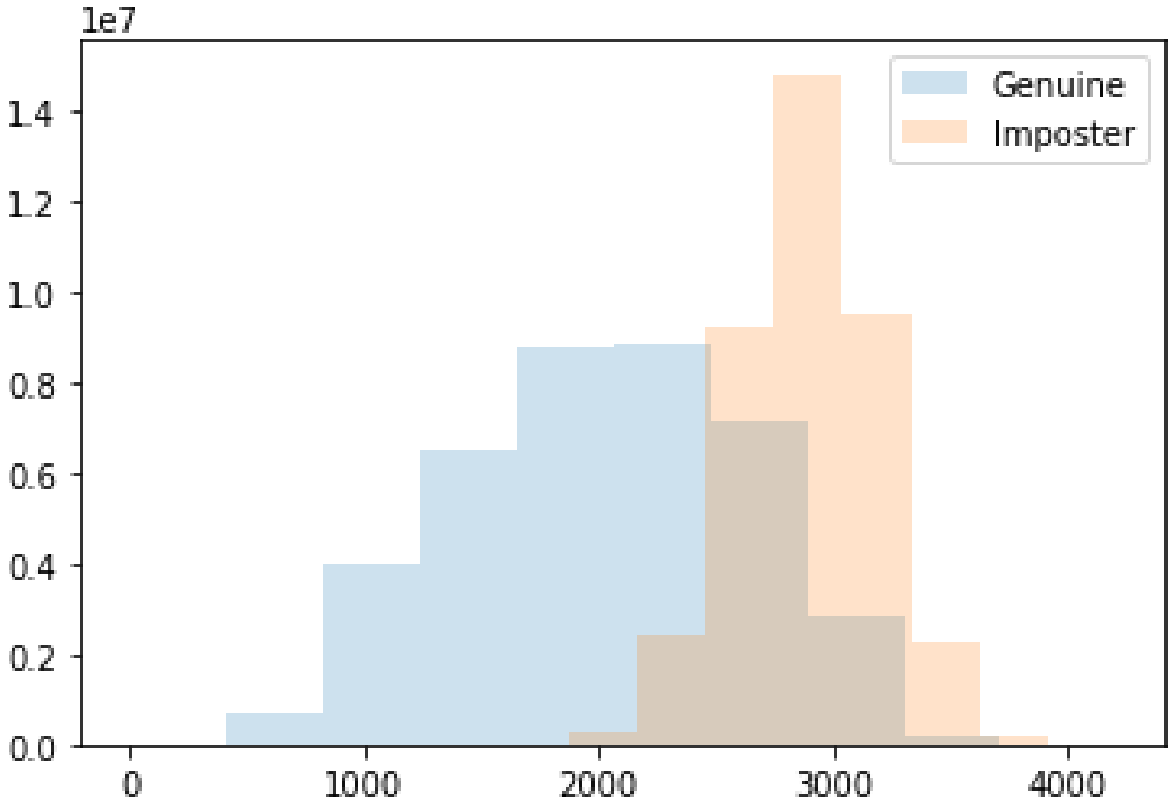

Figure 4: Genuine vs. Imposter Histogram

(f) Generate an ROC curve from the above sets of distances. What is the equal error rate? What is the error rate of a classifier that simply guesses randomly?

For this problem, we first combined our genuine and imposter distance arrays by appending one to the other. We also created an array of 0's with the same length as the genuine array and an array of 1's with the same length as the imposter array. We then appeneded the 0's and 1's array together as well so that the labels corresponded with the combined array for genuine and imposter distances.

We then used the ROC curve function with the array of distances and the array of labels to calculate the false-positive-rate, true-positive-rate, and threshold values. Finally, we plotted the curve using these values to highlight the different tpr-fpr pairs for different threshold values.

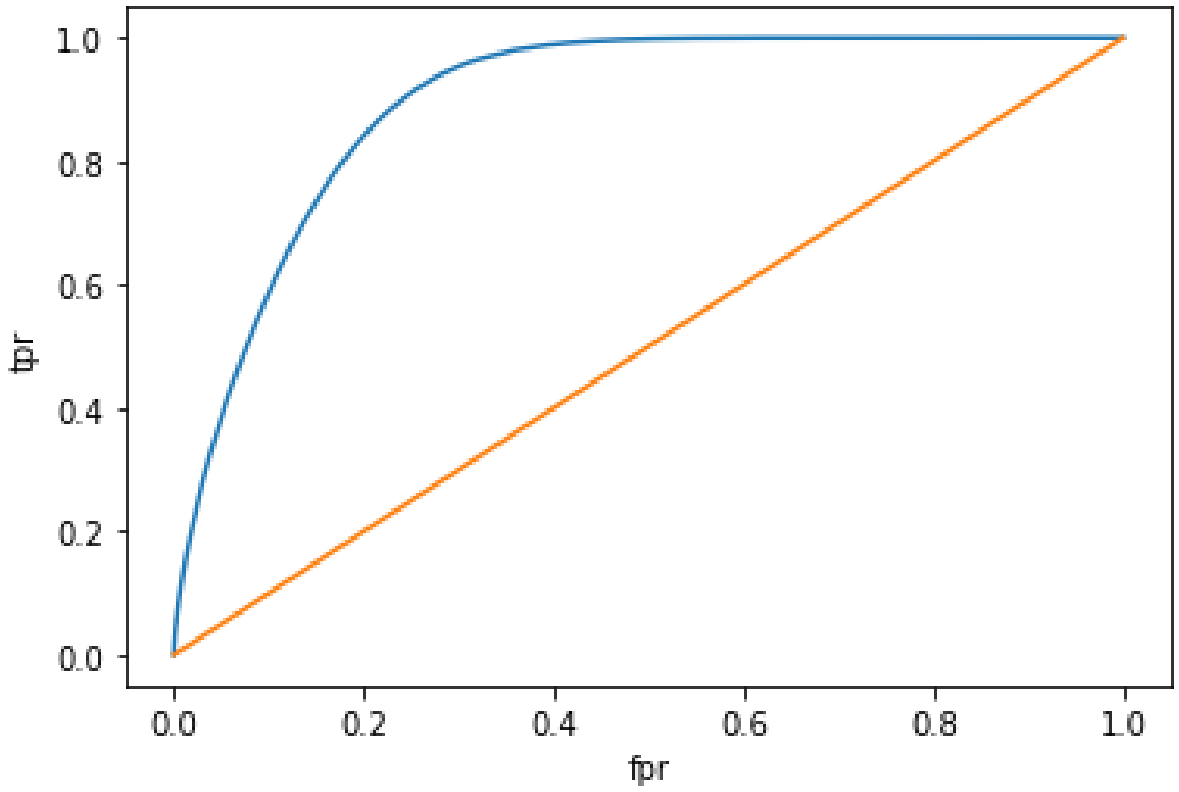

Figure 5: ROC Curve For 0-1 Genuine and Imposter Matches

To calculate the equal error rate, we found the point on the curve at which the difference between fnr and fpr is minimized (or equal to zero). The equal error rate for the curve above is calculated to be **0.185547.**

If the classifier simply guesses randomly, the ROC curve will converge with the linear line plotted from (0,0) to (1,1). This is because a random guess will mean that the false-positive-rate and true-positive rate will be equal, as both are equally likely for guesses. Thus, the EER line (which runs from (0,1) to (1,0)) intersects with the ROC curve at (0.5, 0.5), and so the error rate will be **50 percent** or **0.5** in such a scenario.

(g) Implement a K-NN classifier. (You cannot use external libraries for this question; it should be your own implementation.)

In order to implement our KNN classifier, we first calculated the distance matrix for all points between the train and test set that can be passed into the function. The other two arguments that the function takes in are the k-value and the labels for the train set.

For each test point, the indices for the k smallest distances are identified using the argpartition method. The labels for those indices are then extracted and used to calculate the mode, or most common label, among them. The mode is then the label that is assigned to that test point. Finally, the function returns an array of the predicted label for each test point.

(h) Randomly split the training data into two halves. Train your k-NN classifier on the first half of the data, and test it on the second half, reporting your average accuracy.

To split the training set, we used sklearn to split it so that 50 percent of the training set was the new train set and the other 50 percent would be the test set. We then trained our classifier with the train set using many different k-values and produced predictions for the test points.

Ultimately, we used this train-test split as an opportunity to check our classifier and which k-values produced higher accuracies. We tried many k-values ranging from 1 to 50, but ultimately decided on k = 5, as the accuracy was quite high and reduced computation time.

With k = 5, the average accuracy for our kNN classifier was 95.96 percent, meaning that 95.56 percent of our predicted labels matched the corresponding sample's actual label.

(i) Generate a confusion matrix (of size 10 x 10) from your results. Which digits are particularly tricky to classify?

To generate our confusion matrix, we used a helpful function from sklearn, both for creating the matrix and for visualizing it. However, we altered the visualizing function to produce the visual result we wanted.

As the confusion matrix shows, the majority of the points were predicted to have a label that matched their actual label (these are the counts along the dark red diagonal). However, some digits were particularly tricky to classify based on the confusion matrix's results. For our purposes, we will assume that counts that were incorrectly labeled 40 or more times are particularly tricky – they are: 4's classified as 9's (66 times), 2's classified as 7's (40 times), 8's classified as 3's (43 times), and 9's classified as 7's (44 times). Logically this makes sense – 4's are structurally similar to 9's, 2's are structurally similar to 7's, 8's are structurally similar to 3's, and 9's are structurally similar to 7's. For certain people's handwriting, these could easily be incorrectly classified and our confusion matrix reflects that.

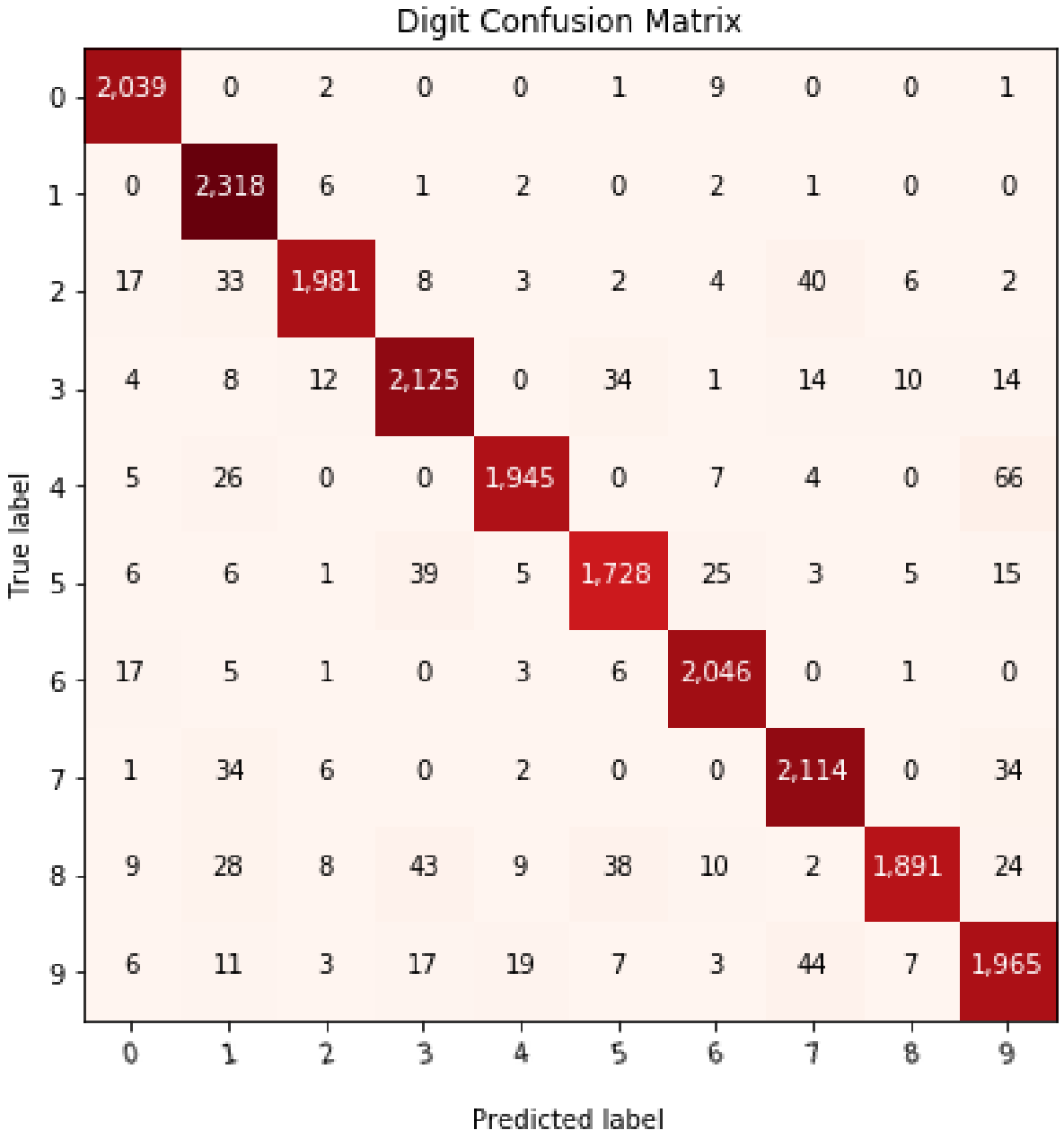

Figure 6: 10 x 10 Confusion Matrix For Digit Labeling

(j) Train your classifier with all of the training data, and test your classifier with the test data. Submit your results to Kaggle.

> For this final part, we trained our classifier on the full training data set with a k value of 5 and exported the predicted labels for the test set to a csv. We then uploaded this csv to the Kaggle website and received an accuracy score of 96.8 percent.

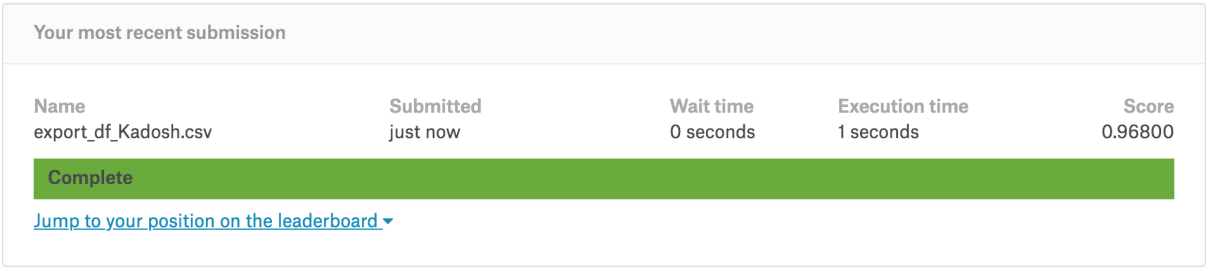

Figure 7: Kaggle Accuracy Score For KNN Classifier

0.3 Problem 2: The Titanic Disaster

(a) Join the Titanic: Machine Learning From Disaster competition on Kaggle. Download the training and test data.

(b) Using logistic regression, try to predict whether a passenger survived the disaster. You can choose the features (or combinations of features) you would like to use or ignore, provided you justify your reasoning.

When determining which combination of features we would use, we applied the approach of first removing the fields we thought were not relevant and then testing those that were relevant in different combinations to better understand how they all influenced the logisitic regression.

Of the variables we could choose to use or ignore (PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked), we removed the following becuase we believed them to be poor indicators: "PassengerID" – It is a unique ID that was assigned to each passenger after the fact, but is not a variable that influenced a passenger's likelihood to have survived, "Survived" – This variable is our label value for training, so it should not be used as a variable in the regression, "Cabin" – While cabin may have been a good indicator, the formatting of the cabin data and the lack of entries made us believe it would not be more useful to the regression than what we could already deduce about a passenger's wealth from their Embarkation, Fare, and Pclass.

For the rest of the variables, we tried different combinations to see which appeared to work best on the training data and then on the test data. We did notice that Sex was a very influential variable in the regression. We also noticed that Age, as it was provided, was not usable. To make Age more usable, we inserted the median age to all NaN cells in the data set. After doing that, we still noticed that age wasn't improving the logistic regression as we had assumed. We realized that the data was not well structured and that age's distribution and values needed to be modified. To improve the regression, we used the standardScalar feature in sklearn to standardize the

data. Once we did that, we were able to bring age back into the combination of features and see an improved score. In addition to Sex and Age, we found that Fare and Sibsp were good indicators of survival. The fare, once standardized, was a strong measure of wealth and which socio-economic class the individual was a part of. We also found that Pclass and fare were conveying similar information, and for purposes of simplicity, we did not use both.

In choosing the combinations of features we wanted to use for the logistic regression, our goal was to use clean data (we cleaned it up if we thought it was missing too many values or needed to be converted into type int), features that we believed would be important and play a role in one's survival on the titanic, and a combination of features that allowed for the highest accuracy score.

Aside from choosing the correct features, the rest of the process was fairly simple. In order to actually test which features and changes were improving our score, we split our training set into train and test subsets, created and fit a logistic regression model, and used the model to predict the test labels and check our accuracy score. Finally, we repeated the same process for our full training set to predict labels for our full test set and exported the CSV to be uploaded to Kaggle (see part c for our Kaggle score).

(c) Train your classifier using all of the training data, and test it using the testing data. Submit your results to Kaggle.

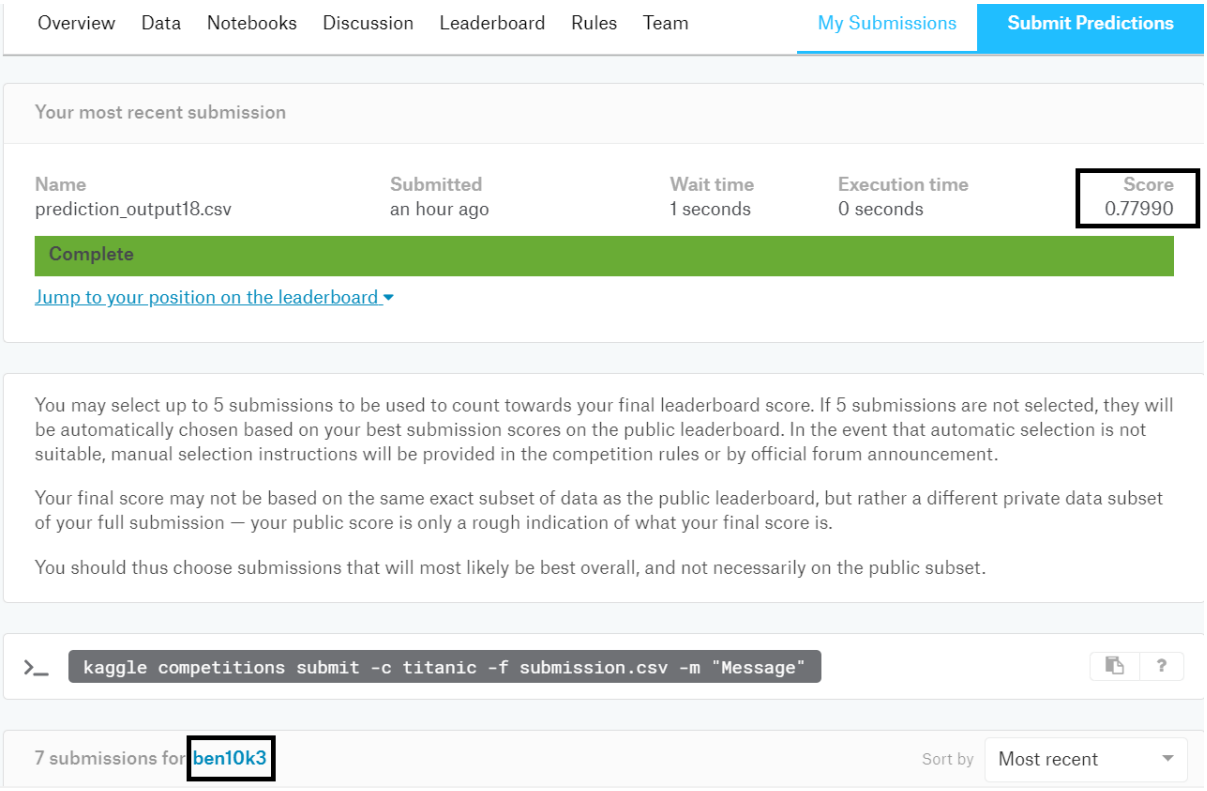

Figure 8: Titanic Submission Results

0.4 Problem 3: Written Exercises

1. Variance of a Difference. Show that the variance of a difference is var[X - Y] = var[X] + var[Y] - 2cov[X,Y], where cov[X,Y] is the covariance between random variables X and Y.

As a first step, we will simplify the equation for covariance given that the cov[X,Y] = $\mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])]$. As a part of this derivation, it is important to recognize that taking the expectation of a constant is equivalent to the constant itself. Therefore, since $\mathbb{E}[X]$ is a constant, $\mathbb{E}[\mathbb{E}[X]]$ would be equivalent to $\mathbb{E}[X]$:

$$\begin{aligned} cov[X,Y] &= \mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])] \\ &= \mathbb{E}[XY - X\mathbb{E}[Y] - Y\mathbb{E}[X] + \mathbb{E}[X]\mathbb{E}[Y]] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}[Y]\mathbb{E}[X] + \mathbb{E}[X]\mathbb{E}[Y] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \end{aligned}$$

Given the following equation for variance, var[X] = $\mathbb{E}[X^2] - (\mathbb{E}[X])^2$, we can begin to derive the equation for var[X - Y]. Throughout this derivation, we make use of the different rules for expectations (adding, multiplying, etc.):

$$\begin{aligned} var[X-Y] &= \mathbb{E}[(X-Y)^2] - (\mathbb{E}[X-Y])^2 \\ &= \mathbb{E}[(X^2 - 2XY + Y^2] - (\mathbb{E}[X-Y])(\mathbb{E}[X-Y]) \\ &= \mathbb{E}[X^2] - 2\mathbb{E}[XY] + \mathbb{E}[Y^2] - (\mathbb{E}[X])^2 + 2\mathbb{E}[X]\mathbb{E}[Y] - (\mathbb{E}[Y])^2 \end{aligned}$$

At this point, we can rearrange our terms in the following way:

$$= \mathbb{E}[X^2] - (\mathbb{E}[X])^2 + \mathbb{E}[Y^2] - (\mathbb{E}[Y])^2 - 2\mathbb{E}[XY] + 2\mathbb{E}[X]\mathbb{E}[Y]$$

Using our formula for variance, we can combine the first two terms to represent the variance of X and the third and fourth terms to represent the variance of Y. We can also factor out -2 in the fifth and sixth terms:

$$= var[X] + var[Y] - 2(\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y])$$

By using the expression that we derived for covariance above, we can ultimately simplify the expression:

$= var[X] + var[Y] - 2cov[X, Y]$

Therefore, it is evident that **var[X - Y] = var[X] + var[Y] - 2cov[X,Y].**

2. Bayes rule for quality control. You're the foreman at a factory making ten million widgets per year. As a quality control step before shipment, you create a detector that tests for defective widgets before sending them to customers. The test is uniformly 95% accurate, meaning that the probability of testing positive given that the widget is defective is 0.95, as is the probability of testing negative given that the widget is not defective. Further, only one in 100,000 widgets is actually defective.

(a) Suppose the test shows that a widget is defective. What are the chances that it's actually defective given the test result?

In order to determine what the chances are that the widget is actually defective given the test result, we will use Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)}$$

In order to solve this problem using Bayes' theorem, we first need to define the variables A and B and determine the corresponding probabilities for each of the terms in the equation.

We can define variables A and B as follows:

A = Defective Widget

B = Positive Test

Given the information provided, we can now calculate the probabilities as follows:

$$P(A) = \frac{1}{100,000}$$

$$P(B|A) = \frac{95}{100}$$

$$P(B|\neg A) = \frac{5}{100}$$

$$P(\neg A) = \frac{99,999}{100,000}$$

Plugging these values into the formula above, we get the following:

$$P(A|B) = \frac{(\frac{95}{100}) * (\frac{1}{100,000})}{(\frac{95}{100}) * (\frac{1}{100,000}) + (\frac{5}{100}) * (\frac{99,999}{100,000})} = 0.0001899$$

Therefore, the chances of the widget being defective, given the positive test result, are **0.0001899** or **0.01899 %.**

(b) If we throw out all widgets that test as defective, how many good widgets are thrown away per year? How many bad widgets are still shipped to customers each year?

Given that 10 million widgets are produced per year, and that only 1 in 100,000 is actually defective, we can determine that of all the widgets produced, 100 will actually be defective.

Now, we can apply the probabilities given to determine how many good widgets are thrown away per year and how many bad widgets still ship to customers.

If there are 100 defective widgets out of 10 million, there are 9,999,900 non-defective widgets. Of those that are not defective, there is a .95 chance that they will test negative. Therefore, there is a .05 chance that non-defective widgets will test as defective:

$$.05 * 9,999,900 = 499,995$$

So, **499,995** good widgets will test as defective and be thrown away.

Regarding bad widgets that will still ship to customers, there are 100 defective widgets, and given that there is a .95 chance that they will test as defective, there is a .05 chance they will not test as defective.

$$.05 * 100 = 5$$

So, **5** bad widgets will test non-defective and still be shipped to customers.

3. In k-nearest neighbors, the classification is achieved by plurality vote in the vicinity of data. Suppose our training data comprises n data points with two classes, each comprising exactly half of the training data, with some overlap between the two classes.

(a) Describe what happens to the average 0-1 prediction error on the training data when the neighbor count k varies from n to 1. (In this case, the prediction for training data point xi includes (xi, yi) as part of the example training data used by kNN.)

In this case, we are taking the points that we used to train the model and then using them as test points. When k = 1, we will look at the closest neighbor to that point. However, since we are using the training points as test points, the closest point will be that exact same point (since it is also present in the training set). Therefore, all test points will be predicted correctly for k = 1 on average because their nearest neighbor will be the exact same point in the training set. In other words, the 0-1 prediction error will be 0.

On the other end of the spectrum, where k = n, we are now testing a point such that the neighbors we will check are all training data points. Since the data is evenly split by class, half of the neighbors are guaranteed to be from one class and half will be from the other class, meaning there is an equal number of neighbors from each class for all cases. Since kNN arbitrarily selects a class in such a tie, we can say that the average 0-1 prediction error is 0.5 because on average, selecting a class from a set of 50 percent class 1 and 50 percent class 2 will be correct half of the time.

For the k values in between 1 and n, the answer is a bit murkier. For those k-values, the distribution of the data, how it overlaps, etc. are all crucial components of how well the kNN model will predict. As you change the k, different neighbors might swing the prediction, but it is highly dependent on the distribution of the data (as this determines which k neighbors will be observed). Therefore, we can say that for k values in between 1 and n, we can expect the average 0-1 prediction error to fluctuate through many different values, starting from a value of 0 at k = 1 to a value of 0.5 at k = n. These fluctuations are not limited to being linear and are not limited to falling between values of 0 and 0.5. A hypothetical example of what this plot could look like is below (many other potential plots exist). The main point is that the error starts at a value of 0.5 for k = n and ends at a value of 0 for k = 1.

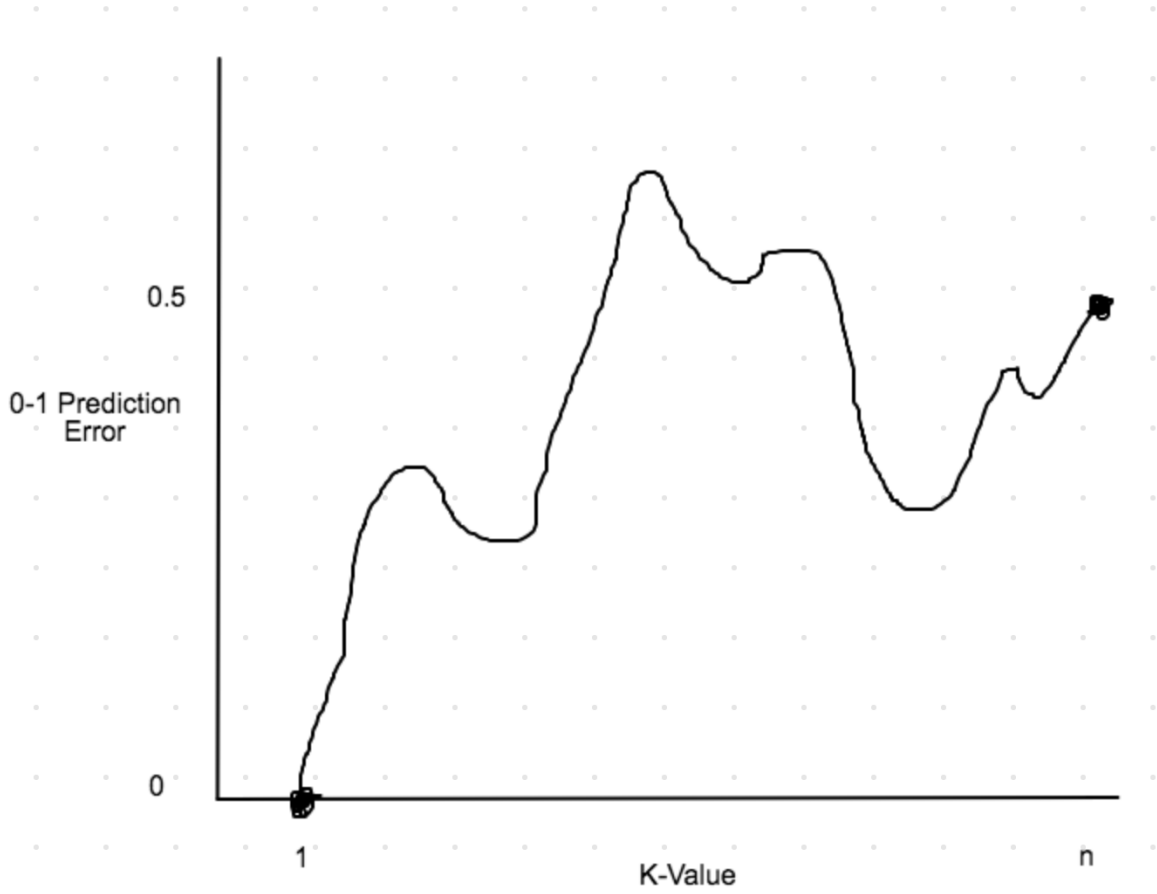

Figure 9: 0-1 Prediction Error vs. K-Value Where Test Set is Train Set

(b) We randomly choose half of the data to be removed from the training data, train on the remaining half, and test on the held-out half. Predict and explain with a sketch how the average 0-1 prediction error on the held-out validation set might change when k varies? Explain your reasoning.

As can be seen in the sketch below, as k varies, so does the variance and bias, which are two sources of prediction error. As the graph shows, there is an optimal k such that the combination of the variance and bias errors leads prediction error to be at a minimum. As k increases, variance decreases. This is because when k is small, the boundaries between classes are more rigid and will change noticeably as k increases until k reaches a more balanced point where there is less variation and the distinction between classes becomes smoother and more clearly defined. However, if k continues to increase and

becomes very large, the distinction between classes starts to blur and overlap such that there is high bias.

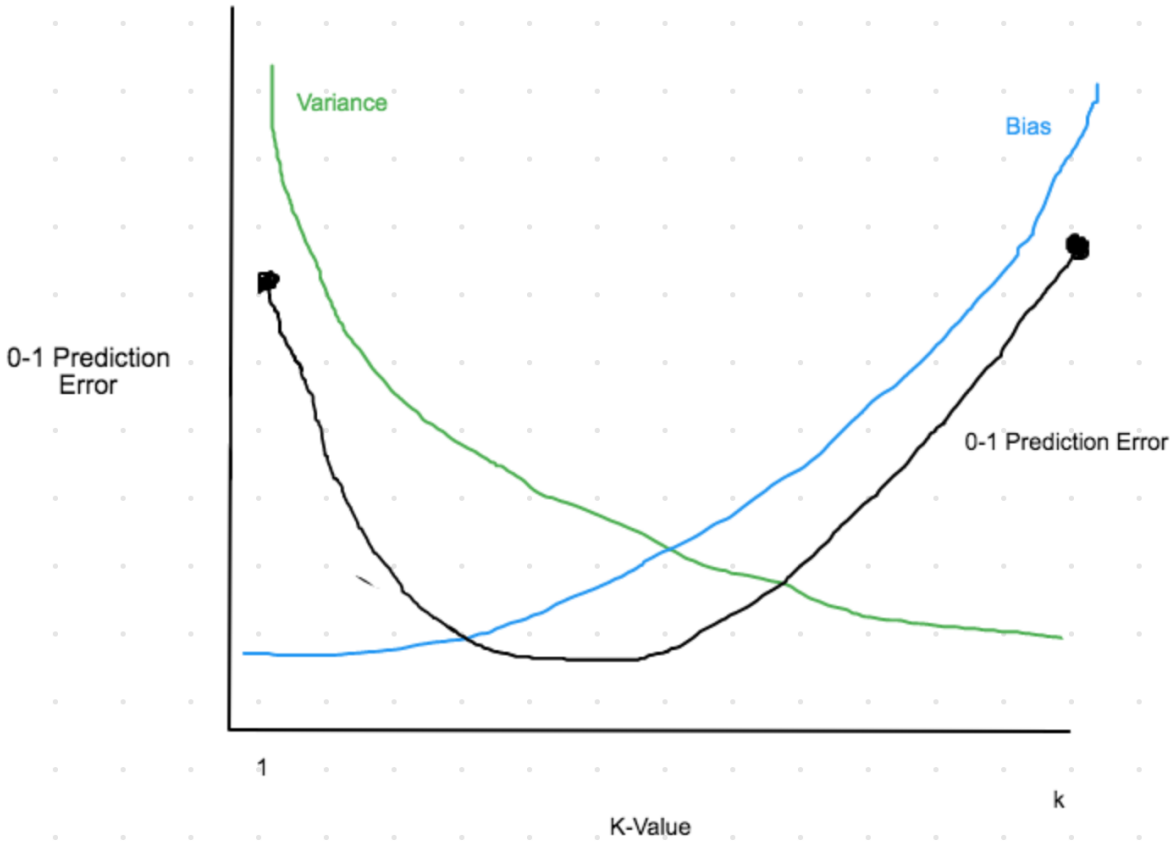

Figure 10: 0-1 Prediction Error vs. K-Value for Validation Set

(c) In kNN, once k is determined, all of the k-nearest neighbors are weighted equally in deciding the class label. This may be inappropriate when k is large. Suggest a modification to the algorithm that avoids this caveat.

A suggested modification would be to use a weighting system that favors/weighs proximity to the data point in question when deciding the class label. To do so, we could calculate the Euclidean distance for all neighbors, and then calculate the inverse of each of those distances to be used as weights to be multiplied to each point when deciding the class label. By taking the inverses, we will be placing more weight on the closer points. For example, if a point is 0.5 units away, its inverse will be 2, and so instead

of just counting its class 1 time as kNN usually does, we would count its class 2 times (multiply 1 by the weight for that point). This ensures that for larger k-values, points that are farther away will not have as much of an influence in the labeling process.

(d) Give two reasons why kNN may be undesirable when the input dimension is high.

There are several reasons why kNN may be undesirable when the input dimension is high, as is pointed out in the textbook. Two of those reasons are:

1. Capturing a percentage of the data to assess the label and proximity of each point within that space requires covering a much larger percentage of the range of each input variable for that data point. Thus, the k closest neighbors may no longer be "neighbors" at all to the given data point. Rather, as the number of dimensions increases, most possible neighbors converge to a very large distance, and therefore are not good predictors for defining the data point in question.

2. When the input dimension is high, all sample points are close to an edge of the sample, so most data points are closer to the boundary of the sample space than to any other data point and prediction is much more difficult near the edges of the training sample.

(e) The plot below shows a binary classification task with two classes: red and blue. Training data points are shown as triangles, and test data points are shown as squares, and all data points are colored according to their class. The line in the plot is the decision rule from training a logistic regression classifier on the training data, with points above the line classified as red and points below the line classified as blue.

i. Calculate confusion matrices for both the training and test data for the logistic regression classifier.

In order to determine the TP, TN, FP, FN counts, we looked at each data point and compared it's color to which predicted color it would be based on the regression. For those above the line, the predicted value would be red and we compared that to what the actual colors were. Similarly, we looked at all points below the regression line, which would be predicted to be blue, and compared each to their actual colors. We also noted whether they were training (triangle) or test (square) data points.

Training

	Actual Red	Actual Blue
Predicted Red	6	3
Predicted Blue	3	8

Test

	Actual Red	Actual Blue
Predicted Red	3	1
Predicted Blue	2	4

ii. Now, calculate confusion matrices for both the training and test data for the kNN algorithm with k = 1 on the same data, using the same train/test split (you should be able to do this by hand).

For this problem, we were interested in populating the confusion matrix using kNN with a k value of 1. For the training data, we looked only at the training points, and looked at the closest neighbor for each training point and determined if the predicted color and the actual color matched or not. We then placed it in the correct location in the matrix accordingly. For the test data, we looked at each point and found its closest neighbor among the training points and compared it's actual color to that of its closest neighbor. If they were the same color, then the predicted and actual matched; otherwise, they did not match and we populated the matrix accordingly.

Training

	Actual Red	Actual Blue
Predicted Red	5	5
Predicted Blue	4	6

Test

	Actual Red	Actual Blue
Predicted Red	1	2
Predicted Blue	4	3

iii. Discuss the difference in accuracy on both the training and test data between the two algorithms. Explain why this difference occurs.

We used the formula, accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$, in order to calculate the accuracy values. The accuracy for the training and testing data for each of the two algorithms is as follows:

Accuracy	Training	Test
Algorithm 1: Logistic Regression	14/20	7/10
Algorithm 2: kNN (k = 1)	11/20	4/10

As we can see, the accuracy for Algorithm 1 is 70 percent for both the training and test data, while the accuracy for Algorithm 2 is between 40 to 55 percent for the training and test data. This is because the logistic regression line has a much better fit and has taken into account all of the other data points to dictate its fit, whereas for kNN with k=1 there is a very high degree of variability, and thus the accuracy rate is lower. The logistic regression algorithm, however, has a fit that is more optimized to the data.

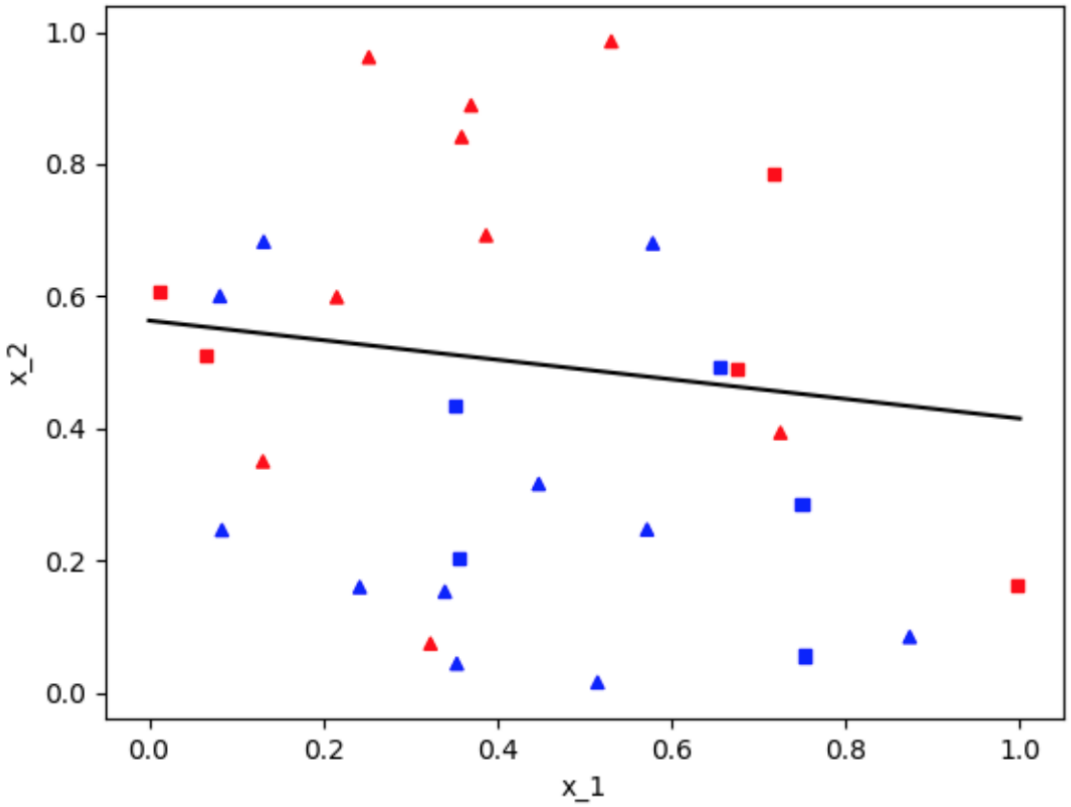

Figure 11: Training/Test Data Plot