Chad Pickering 913328497

11/3/2015

A03

Corresponded with: Rico Lin, Ricky Safran, Sierra Tevlin, Janice Luong, Hannah Kosinovsky

Resources: Stack Overflow, Office Hours, Piazza Forums, Chris Murphy, Randtests package documentation

STA141 Assignment 3: K Nearest Neighbors and Cross Validation

Premise: We are given 5000 images with 28x28 pixels each, each representing an image of a handwritten digit. These should be classified as the digit they represent as accurately as possible using the K Nearest Neighbors algorithm and 5-fold cross validation. The code for drawing each image was given, and will be used in the first phase.

Exploratory Phase: In order to get familiar with the images and their overall patterns and behavior, I found how many pixels (and which pixels) over all 5000 images are never used (have a maximum value of 0). 131 out of the 784 are white and never used – most of the area around the border of the image space. Using a similar method, I found that no pixel is used at 255 constantly, which is to be expected with 5000 members of the sample. Further on, I decided to split the images by digit and draw the first 100 of each to see a healthy distribution of how handwriting varies and what to expect later on.

Interestingly, when drawing the means of each digit type (0, 1,...,9) and the grand mean of all 5000 images, I was reminded of ANOVA, where each digit type is a treatment and we could (and eventually would in distance to average, essentially) study the variation in deviations from the average between digits and within specific digits. When drawing the standard deviation of each digit "level," I could identify where certain handwriting patterns are more frequent, such as crossing the 7 or making the 9 more g-like. The slant of the 1 has perhaps the most variance, but because there are more right-handed people in the population, and in the sample, there is more of a tendency of a right-tilted 1. As expected, in the standard deviation, though, there is almost no variance where most 1's intersect right in the middle. These identifiers and patterns may make classifying the digits much more manageable, in that unique variabilities or attributes that set a digit clearly apart from the others may be used to quickly identify them if all or most share that attribute. In my solution, these are not directly sought after, but used in general in kNN.

Finally, just to test if the frequency of each digit is uniform, I used a chi-squared goodness of fit test. The test statistic is 43.58 and the p-value is essentially zero, meaning that the null hypothesis is rejected and I can conclude that the frequencies of the digits are not uniform. This tells me that some digits appear more often than others well beyond statistical chance.

Randomization and Data Collection Considerations: Upon exploring the data in R and visually in the .csv file, I realized that the order in which the images are organized are not strictly all 0's first, and then all 1's next, and so on. If this was the case, both the kNN and cross validation processes would be greatly affected and results would be biased. For example, if only 0's and 1's are in the first fold, it will more accurately predict 0's and 1's from the test set, especially because of their stark visual differences. In contrast, the order in which the labels are in the data frame already appear random. I did consider other ordering techniques that could have been done, such

as by human subject or location/zip code. This being said, the only way to statistically conclude that a string of digits are in a random order, such as the digits in pi or root 2 (which they are), is the non-parametric runs test. Running this test in R gives 2255 runs, resulting in a test statistic of 0.0236 and a p-value of 0.9812, giving strong evidence that the labels are already in a random order (fail to reject Ho). If a string of digits is truly random, you would expect each digit to be independent of the last, and so there is always a 10% chance that the same digit appears consecutively. So we should not expect digits to always change continuously in the sequence, and we should also not expect the same digit to be repeated multiple times in a row; there should be a healthy mixture of events.

Metrics: To compute the distances between the images, I used three metrics, Euclidean, or straight-line distance (hypotenuse), Manhattan, or "taxicab" distance (adjacent and opposite), and Canberra, or weighted Manhattan. These are 5000 x 5000 matrices, with the 784-dimensional distances between each image in each cell, and zeroes down the diagonal.

K-Nearest Neighbors / Cross Validation Process (#1): The test and training sets in each fold are subsetted out of the original distance matrices in the first function. In the kNN function, the user inputs the distance matrix fold from the train function, a k value, the distance metric, and the original dataset. This function will pass in a row from the test set (1 to 1000), generate the distances to the members of the training set and subset these distances from 1 to k (meaning the k smallest). The function ultimately returns a single prediction. The kNN function is called for every image in the test set in the next function, which repeats the kNN process for each fold. Next, I find the misclassification rates for each k for a single metric that I specify. See the comments in my code for detailed explanations of each line. Ultimately, the best metric/k-value pairs were:

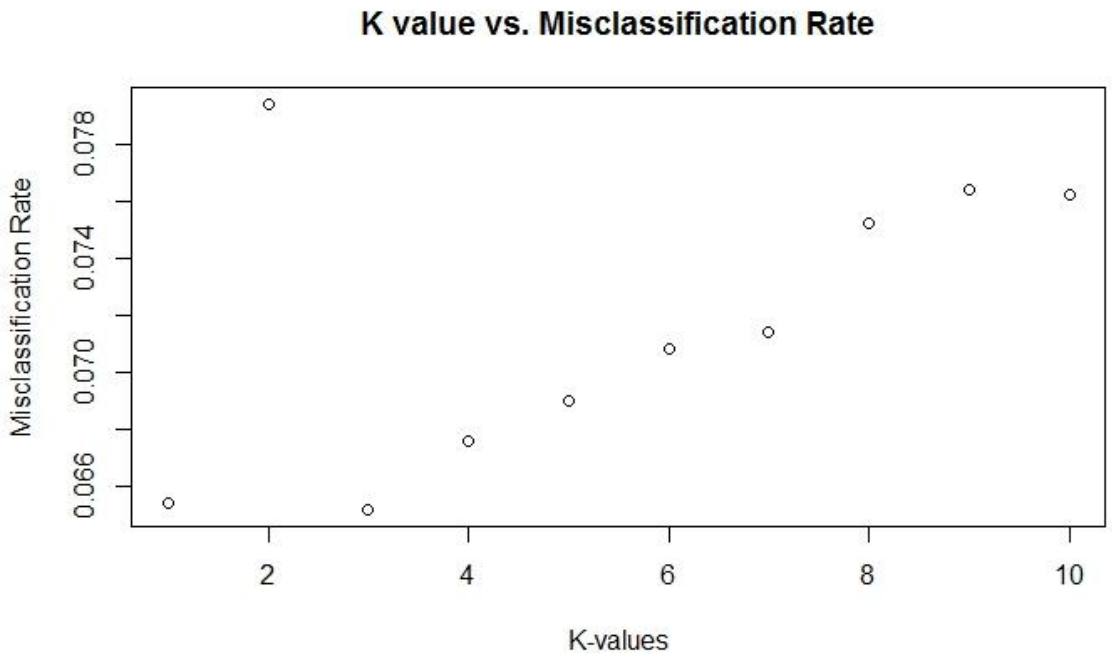
Metric	K-value	Misclassification Rate
Euclidean	3	0.0652
Manhattan	1	0.0762
Canberra	6, 7	0.0678

The best model (lowest misclassification rate) is Euclidean k=3.

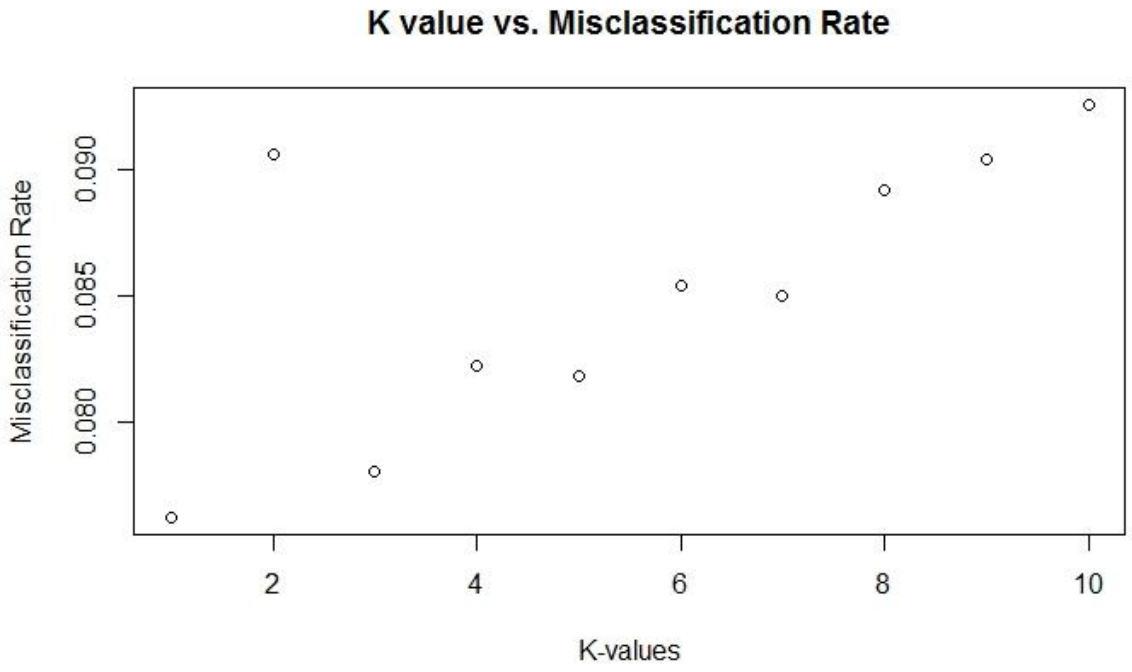
Breaking Ties: I did not account for ties, i.e. code for any modifications. If there is a tie, R chooses the lower number option (i.e. chooses to pick digit 3 over digit 7 if the frequencies are tied), therefore creating a small bias towards lower digit values. However, ties are very uncommon, so the bias should not be statistically significant.

Plots of K vs. Misclassification Rate (#2): The following are the plots of the first 10 k values for each metric. If zoomed to k=100 or more, you can see the general $x^{1/2}$-like trend.

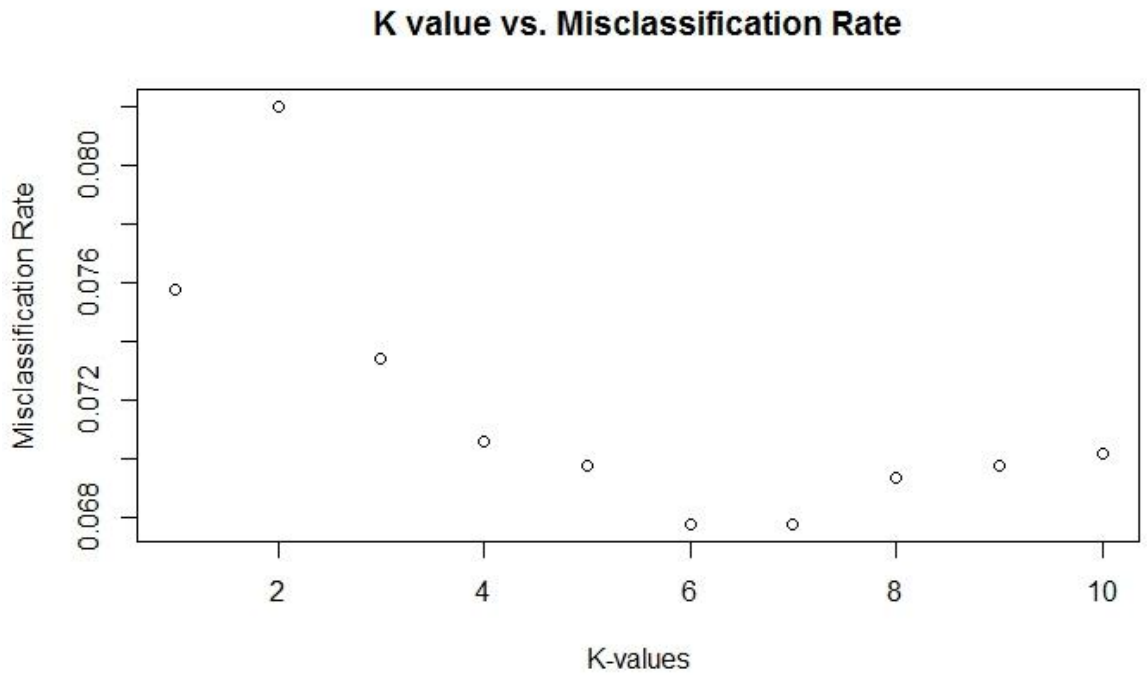
EUCLIDEAN:



MANHATTAN:



CANBERRA:



Confusion Matrix (#3):

The following is the confusion matrix for Euclidean k=3. The row indices represent the actual digits that the images truly represent, and the column indices represent the digit value that an image was classified as using kNN. The diagonal is populated significantly more than everywhere else because the misclassification rate for the model was rather low. An example of how to read the matrix: 12 actual 2's were misclassified as 7's, and 500 3's were classified correctly.

	0	1	2	3	4	5	6	7	8	9
0	487	0	0	0	0	0	3	0	0	0
1	0	586	1	1	2	0	1	2	0	1
2	5	17	467	4	2	1	1	12	3	2
3	2	3	5	500	1	8	0	1	0	3
4	1	11	2	0	432	0	2	2	0	26
5	6	3	0	20	1	404	6	0	3	5
6	6	3	1	0	2	2	480	0	2	0
7	0	14	2	1	2	1	0	524	0	10
8	4	10	4	16	2	17	1	4	357	11
9	4	3	3	6	16	0	0	9	1	437

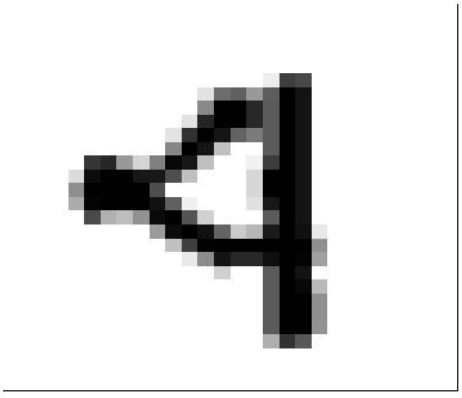
Classified Best / Worst (#4): 0 is classified the best with a 487/490 success rate. 8 is classified worst by far with a 357/426 success rate. 8 generally has more pixels with non-zero values, and therefore has a greater probability of sharing non-zero pixels with other digits and being confused with them.

	numbers	correct_prop
1	0	0.9938776
2	1	0.9865320
3	2	0.9085603
4	3	0.9560229
5	4	0.9075630
6	5	0.9017857
7	6	0.9677419
8	7	0.9458484
9	8	0.8380282
10	9	0.9123173

Digits Generally Confused (#5): Based on the confusion matrix, 2's and 8's were generally confused with all other digits, as they are the only digits without any zeroes in a column. Some maximum non-diagonal values were: actual 5's were predicted as 3's 20 times; actual 4's were predicted as 9's 26 times.

Digits That Are Misclassified / Difficult for Human to Classify (#6): Upon looking at the non-diagonal elements of the confusion matrix, I realized that the largest value, as reported above, was 26 4's predicted as 9's. I found the union of "Actual 4" (event A) AND "Predicted 9" (event B); which is 26 indices, and then used the draw() function to view all 26. I have an example below of the most unusual member of the group.

Something to understand: when a digit is misclassified, it does not necessarily look like a different number to us. In other words, it may be understandable to the human eye but to the machine, it was closer in pixel values to the digit the program classified it as.



Distance to Average: The general premise of this is to calculate the average value of each individual pixel for each digit and then calculate the distance between the grand average for each digit and each individual image. This involves attaching the 784 average pixel values for each digit to the matrix from part 1, essentially, and performing an analogous process. This means that each of the 5000 images are "compared" to the 10 averages, and whichever is "closest" (has least distance between) wins and is assigned that digit's classification. This method does not weight the averages or compensate for extreme outliers, so the misclassification rates are substantially higher than those of the kNN method.

Metric	**Distance to Average**	**K Nearest Neighbors / CV**
Euclidean	0.1848	0.0652
Manhattan	0.3386	0.0762
Canberra	0.4502	0.0678