Pattern Recognition Laboratories

Project 6

EE 516 - 01

Pattern Recognition

Spring 2023

Group 4

Students: Colt Whitley, Kai Ponting, Nathan Jaggers

Project Due: 06/14/23

Instructor: Dr. Zhang

K-Nearest Neighbor Classifier for Handwritten Digits

Assignment

In this project, you will perform the handwritten digit classification of 0 and 1 first, then digit classification of 0 and 2, by using k-nearest neighbor classifier. Experiment with k values of 1, 3, 5 and 11.

You will also experiment with different features, first using the best two features selected from Project 1, then using all raw pixel values (28x28). Again, training sets are used for training and the test sets for test. Report the performance on the test data in terms of error rate and run time for both the Euclidean distance measure and the city-block distance measure.

Procedure

This project was able to build off of work done in project 3 to preprocess and extract features from the MNIST handwritten digits dataset. With this baseline we constructed two types of feature matrices: one composed of extracted features such as circularity and eccentricity, and one that treated each individual feature vector as a feature. In the former case our feature vectors each had only two elements, and in the latter each feature vector held 784 features corresponding to each pixel in the digit. This process was repeated for test data supplied from MNIST such that feature matrices for training and test were easily accessible for both two feature and 784 feature cases. The process was repeated again for the zero-two data set.

Once features were extracted we created two functions, one to measure Minkowski distance, and one to find the k nearest neighbors. Exact code implementations can be found in the appendix, but this functional decomposition allows for increased code flexibility as changing the distance measurement type and number of neighbors is as simple as changing parameters

passed to a function. The Minkowski distance measurement function implemented equation 58 from chapter 4 of *Pattern Classification* by Richard O. Duda, Peter E. Hart, and David G. Stork. The k nearest neighbor function used the Minkowski distance measurement to output the distance from the test vector to all vectors in the training data. If the distance to a training vector is less than any vectors already on the list of k closest vectors, the new training vector is inserted into the list, the new farthest on the list is deleted, and the label of the new vector is recorded. After this process has been completed for all training vectors the mode of the k nearest label vector is taken and assigned to the test vector. After this has been completed for all test vectors we compared the test vector labels to the classified test vector labels and calculated the error. This process was repeated for k values of 1, 3, 5, 11 and for both city block and euclidean distance measurement.

While classifying all the training data from all the various feature matrices, the code was also timing how long each classification took, and then computing the per vector time based on the number of test vectors. Finally both the error rates and code timing are output to the console so they can be collected and analyzed in the report.

Results

Table 1: Summary of Error Rates for KNN Iterations

			Error Percentage [%]	
Input Features	Distance Type	K Value	0 and 1	0 and 2
Two Features	City Block	1	1.82	9.27
		3	1.31	7.27
		5	1.26	6.78
		11	1.16	6.88
	Euclidian	1	1.82	9.37
		3	1.31	7.13
		5	1.26	6.98
		11	1.16	6.78
All Pixels as Features	City Block	1	0.10	0.90
		3	0.15	0.85
		5	0.15	0.85
		11	0.15	1.00
	Euclidian	1	0.10	0.90
		3	0.15	0.85
		5	0.15	0.85
		11	0.15	1.00

The first part of Table 1 and Figures 1 and 2 show the error rate for the classifier for two extracted features from the dataset. As can be seen from these figures, there is a general downward trend of error percentage as the number of nearest neighbors is increased. However, as

the number of nearest neighbors goes past 3, the rate of this decrease flattens out significantly, indicating that there is little benefit in counting more than 3-5 nearest neighbors.

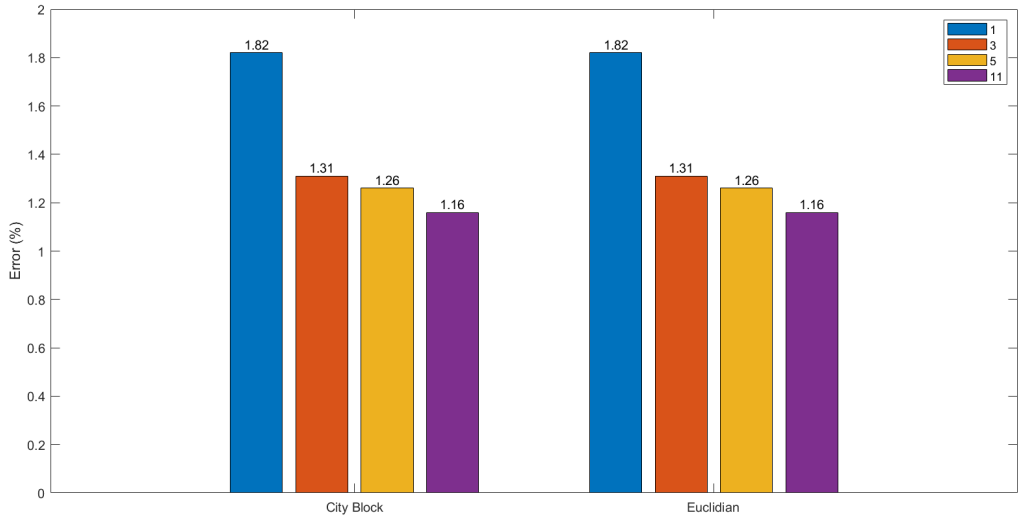

Figure 1: Two Feature Error for 0 and 1 Classification

This pattern occurs for both the 0 and 1 classification and the 0 and 2 classification. The error rates between zero and two are significantly higher, likely due to the fact that the original feature datasets from the two features for 0 and 2 were not as separated as the features selected for 0 and 1. This leads to a higher error rate overall as can be seen in Table 1 and Figure 2.

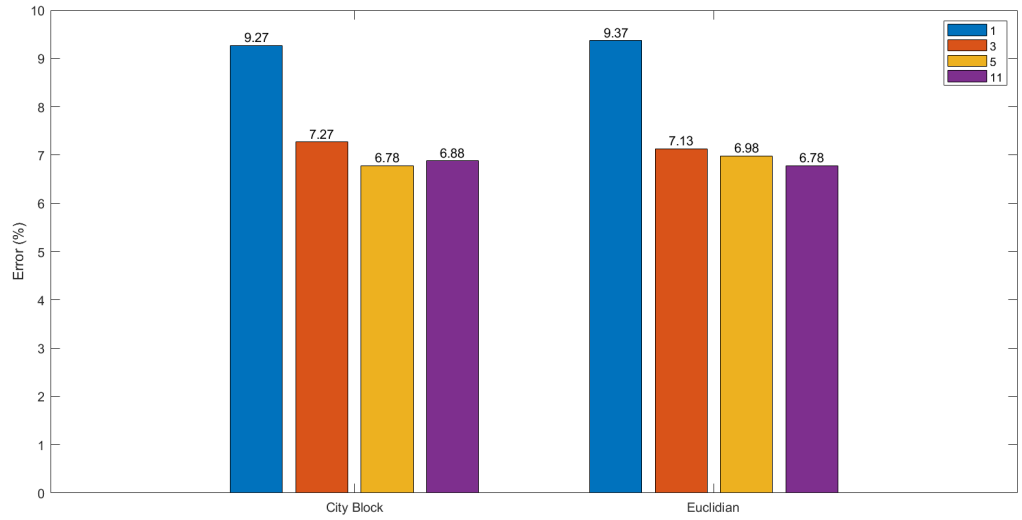

Figure 2: Two Feature Error for 0 and 2 Classification

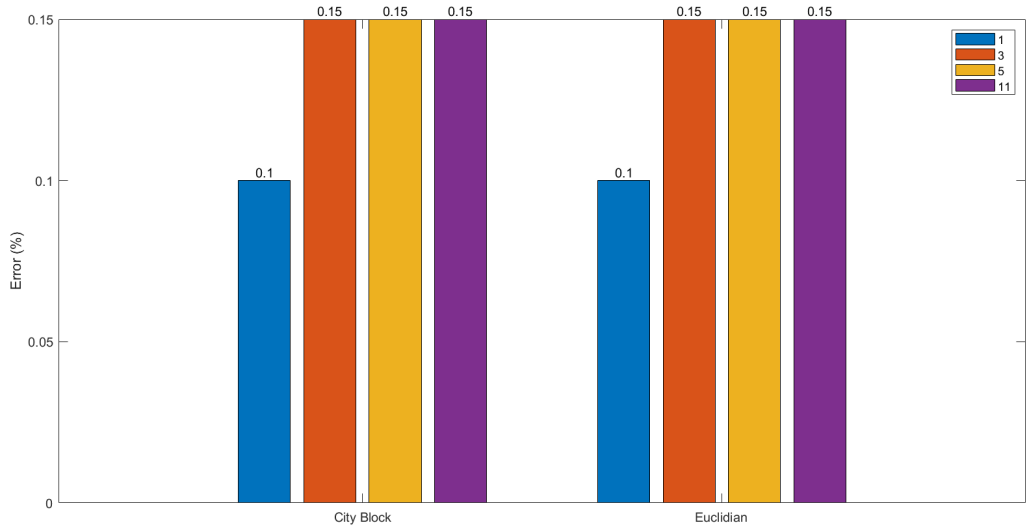

Figure 3: All Pixels Feature Error for 0 and 1 Classification

The second part of Table 1 and Figures 3 and 4 show the error rate of the classifier that uses all individual pixels as features. Using this classifier, we see that there is a very small difference in error between the different nearest-neighbor counts. This is likely due to the fact

that there are so many features, adding more neighbors does not increase accuracy when there are many features that are already very separable as-is.

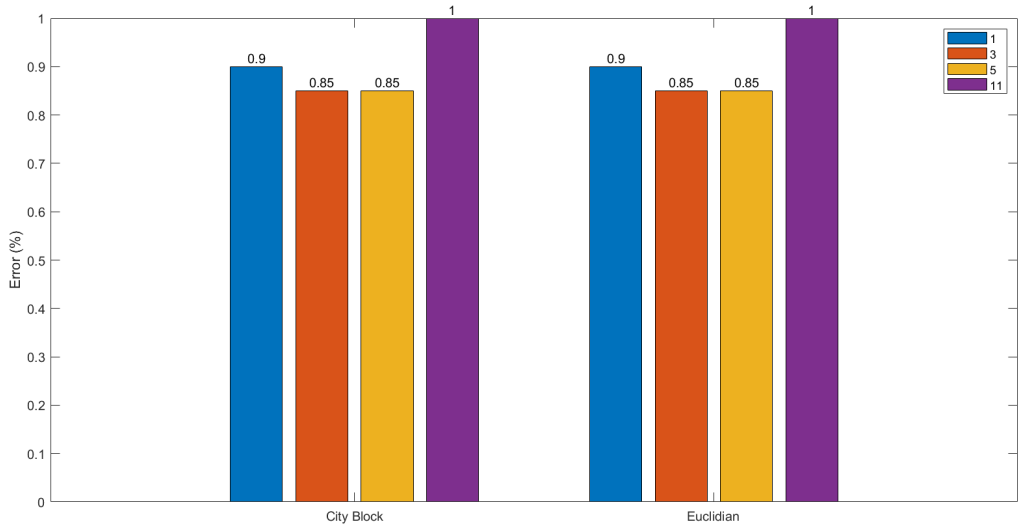

Figure 4: All Pixels Feature Error for 0 and 2 Classification

We see in comparing Figure 3 and Figure 4 that the error rate is on average around 1% for the classification between 0 and 2, whereas it is around 0.12% for the classification between 0 and 1. This is likely due to the fact that 1 and 0 have very different shapes, regardless of translation, rotation, or thickness. This is not the case for 0 and 2, which are much more similar in terms of shape and size.

Table 2: Summary of Simulation Times for KNN Iterations

			Simulation Time [ms]			
Input Features	Distance Type	K Value	0 and 1		0 and 2	
			Per Sample	Total	Per Sample	Total
		1	27.06	53,625	25.49	51,155

Two Features	City Block	3	27.00	53,516	25.17	50,517
		5	27.59	53,677	25.44	51,065
		11	27.50	54,507	25.82	51,829
	Euclidian	1	27.33	54,169	25.44	51,059
		3	27.47	54,437	25.33	50,846
		5	27.18	53,867	25.44	51,061
		11	27.64	54,781	25.83	51,845
All Pixels as Features	City Block	1	139.6	276,743	122.0	244,922
		3	137.8	273,491	124.0	248,798
		5	132.7	263,203	124.2	249,247
		11	131.9	261,453	124.7	250,264
	Euclidian	1	129.6	256,952	118.7	238,301
		3	127.7	253,322	120.6	242,101
		5	128.8	255,192	123.4	247,680
		11	129.1	255,928	121.4	243,651

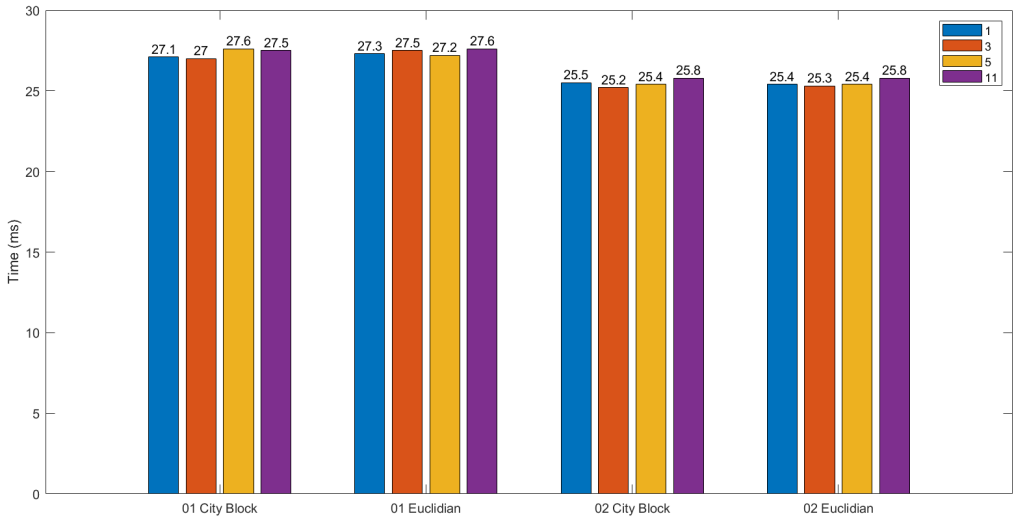

Figure 5: Two Feature Runtime per Sample

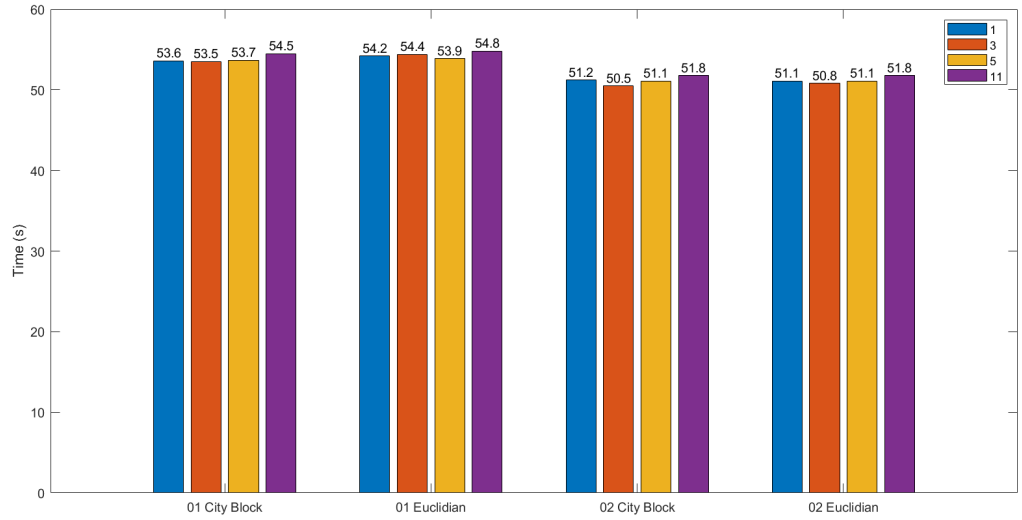

Figure 5: Two Feature Runtime Total

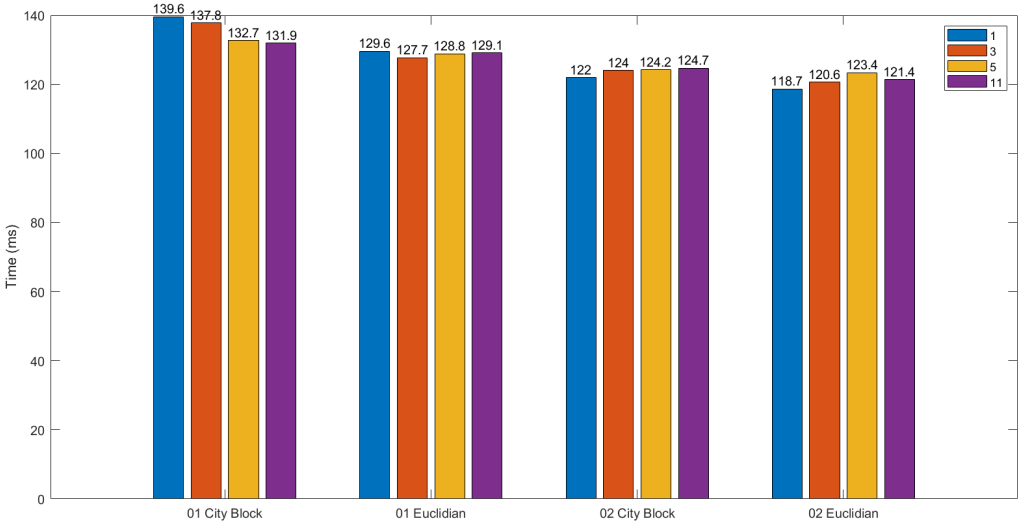

Figure 6: All Pixels Feature Runtime per Sample

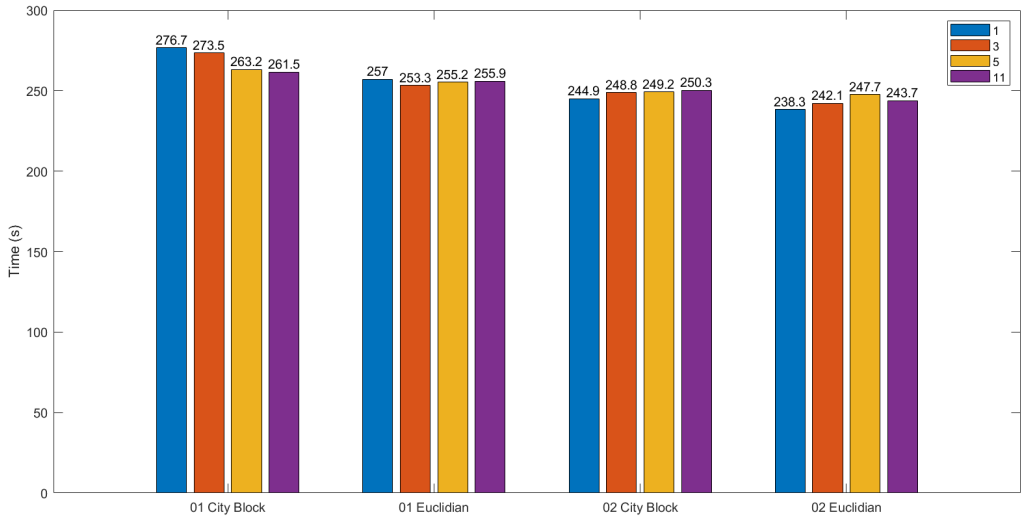

Figure 7: All Pixels Feature Runtime Total

From the plots and Table 2 above, it is very clear to see that changing the number of nearest neighbors does not significantly increase the runtime. This is likely due to the fact that the majority of the time is spent calculating the distance of the sample to every other point, and the comparisons that are made to create k-nearest neighbors is a small fraction of the total time and only increased the per-sample operation time a little bit. We can see this difference in the per-sample time, where higher k values result in around an 0.5 ms increase in time per sample.

We can also see a significant increase in the time taken both per sample and overall by the all pixels as features classifier and the 2-feature classifier. This is due to the large amount of points associated with the all pixels classifier greatly increasing the time needed to perform classification. This is one disadvantage of a KNN classifier, as all of the points used to create the classifier need to be used to perform classification.

In summary, the KNN classifier designed for this project used both 2 features extracted from the dataset as well as using all pixels of each digit as pixels. The all pixels as features

classifier had lower error rates than the 2-feature classifier, but also took a proportionally longer time to analyze the data given that it contained a much larger number of points.

Reflection

Colt Whitley

I found this project quite enjoyable due to its ease of implementation and impressive results. The intuition with the K-nearest-neighbor algorithm makes sense and being able to get results accurate to less than 1% error really shows how powerful it is. Additionally I felt that the long run time for this lab really illustrates the tradeoff between power and speed. While I was not surprised the 28 x 28 pixel performed better I was surprised by how much better it performed. Overall I think this lab did a great job illustrating how it is possible to have successful classifiers without understanding the underlying distributions.

Kai Ponting

It was interesting to learn about the KNN classifier and see how such a simple classifier can perform on digit recognition. Seeing how good performance also is very costly in terms of computational time was interesting as well. In the future, I would want to investigate how this could perform on the Iris and breast cancer datasets that we looked at in Project 5, as I think KNN would be well-suited to this task.

Nathan Jaggers

This lab we explored a classification problem that we had previously seen in Project 3 but with a new approach. This time around we use the K-Nearest Neighbor to classify our data. In lecture we had talked about the pro of KNN being that it was easy to understand but computationally expensive. This Project is a great example of how long KNN can take when dealing with two or more features and a sizable data set. It was interesting to see how in several

cases, we had error increase when increasing K and not conforming to my initial expectations. I have since learned that this may be improved with a larger dataset. This has been a very enlightening project.

Teamwork

Colt Whitley

Wrote MATLAB Code, Data section Tables, Introduction and Procedure.

Kai Ponting

Proofread and wrote interpretation paragraphs for the results section.

Nathan Jaggers

Discussed and compared strategies with Colt for how to construct the classifier. Took data Colt created and made the bar graphs.

Appendix

Full Project Script

```
%% EE 516 Project 6
close all; clc; clear;
%% Load and Preproccess Images into Bianary
% Load Images from files
ims0 = imread("Images/mnist_train0.jpg");
ims1 = imread("Images/mnist_train1.jpg");
ims2 = imread("Images/mnist_train2.jpg");
ims0_test = imread("Images/mnist_test0.jpg");
ims1_test = imread("Images/mnist_test1.jpg");
ims2_test = imread("Images/mnist_test2.jpg");
% Convert to bianary images
bw0 = ims0 > 100;
bw1 = ims1 > 100;
bw2 = ims2 > 100;
bw0_test = ims0_test > 100;
bw1_test = ims1_test > 100;
bw2_test = ims2_test > 100;
% Splice image into individual digits
num0s = 5923;
num1s = 6742;
num2s = 5958;
num0s_test = 980;
num1s_test = 1002;
num2s_test = 1027;
%Width and height of each primary image
width0 = length(bw0(1, :)) / 28;
height0 = length(bw0(:, 1)) / 28;
width1 = length(bw1(1, :)) / 28;
height1 = length(bw1(:, 1)) / 28;
width2 = length(bw2(1, :)) / 28;
height2 = length(bw2(:, 1)) / 28;
width0_test = length(bw0_test(1, :)) / 28;
height0_test = length(bw0_test(:, 1)) / 28;
width1_test = length(bw1_test(1, :)) / 28;
height1_test = length(bw1_test(:, 1)) / 28;
width2_test = length(bw2_test(1, :)) / 28;
height2_test = length(bw2_test(:, 1)) / 28;
%initialize 3d matricies for storing individal digits
bw0seperated = zeros(28,28,width0*height0);
bw1seperated = zeros(28,28,width1*height1);
bw2seperated = zeros(28,28,width2*height2);
bw0seperated_test = zeros(28,28,width0_test * height0_test);
bw1seperated_test = zeros(28,28,width1_test * height1_test);
bw2seperated_test = zeros(28,28,width2_test * height2_test);
% add each picture to a 3d array where each slice is a 28 by 28 image
for i = 0:(width0-1)
   for j = 0:(height0-1)
```

```
        bw0seperated(:,:,sub2ind([width0 height0], (i + 1), (j + 1))) = ...
        bw0(((1 + i * 28):(28 + i * 28)), (1 + j * 28):(28 + j * 28));
    end
end
%Remove all empty images of 0s
bw0seperated = bw0seperated(:,:,1:num0s);
% add each picture to a 3d array where each slice is a 28 by 28 image
for i = 0:(width1-1)
    for j = 0:(height1-1)
        bw1seperated(:,:,sub2ind([height1 width1], (j + 1), (i + 1))) = ...
        bw1(((1 + j * 28):(28 + j * 28)), (1 + i * 28):(28 + i * 28));
    end
end
%Remove all empty images of 1s
bw1seperated = bw1seperated(:,:,1:num1s);
% add each picture to a 3d array where each slice is a 28 by 28 image
for i = 0:(width2-1)
    for j = 0:(height2-1)
        bw2seperated(:,:,sub2ind([height2 width2], (j + 1), (i + 1))) = ...
        bw2(((1 + j * 28):(28 + j * 28)), (1 + i * 28):(28 + i * 28));
    end
end
%Remove all empty images of 2s
bw2seperated = bw2seperated(:,:,1:num2s);
% add each picture to a 3d array where each slice is a 28 by 28 image
for i = 0:(width0_test-1)
    for j = 0:(height0_test-1)
        bw0seperated_test(:,:,sub2ind([height0_test width0_test], (j + 1), (i + 1)))
= ...
        bw0_test(((1 + j * 28):(28 + j * 28)), (1 + i * 28):(28 + i * 28));
    end
end
%Remove all empty images of 2s from the test data
bw0seperated_test = bw0seperated_test(:,:,1:num0s_test);
% add each picture to a 3d array where each slice is a 28 by 28 image
for i = 0:(width1_test-1)
    for j = 0:(height1_test-1)
        bw1seperated_test(:,:,sub2ind([height1_test width1_test], (j + 1), (i + 1)))
= ...
        bw1_test(((1 + j * 28):(28 + j * 28)), (1 + i * 28):(28 + i * 28));
    end
end
%Remove all empty images of 2s from the test data
bw1seperated_test = bw1seperated_test(:,:,1:num1s_test);
% add each picture to a 3d array where each slice is a 28 by 28 image
for i = 0:(width2_test-1)
    for j = 0:(height2_test-1)
        bw2seperated_test(:,:,sub2ind([height2_test width2_test], (j + 1), (i + 1)))
= ...
        bw2_test(((1 + j * 28):(28 + j * 28)), (1 + i * 28):(28 + i * 28));
    end
end
%Remove all empty images of 2s from the test data
bw2seperated_test = bw2seperated_test(:,:,1:num2s_test);
```

```
%Release unused vectors
clear width0 width0_test width1 width1_test width2 width2_test;
clear height0 height0_test height1 height1_test height2 height2_test;
%% Extract 0's Features
%Initialize vectors to zero
circularity0 = zeros(num0s, 1);
eccentricity0 = zeros(num0s, 1);
ConvexArea0 = zeros(num0s, 1);
perimeter0 = zeros(num0s, 1);
%Use matlab's built in function to extract features
for i = 1:num0s
   stats = regionprops(bw0seperated(:,:,i), 'Circularity', ...
                       'Eccentricity', 'ConvexArea', 'Perimeter');
   circularity0(i) = stats.Circularity;
   eccentricity0(i) = stats.Eccentricity;
   ConvexArea0(i) = stats.ConvexArea;
   perimeter0(i) = stats.Perimeter;
end
%% Extract 1's Feature
%Initialize vectors to zero
eccentricity1 = zeros(num1s, 1);
ConvexArea1 = zeros(num1s, 1);
%Use matlab's built in function to extract features
for i = 1:num1s
   stats = regionprops(bw1seperated(:,:,i),'Eccentricity', 'ConvexArea');
   eccentricity1(i) = stats.Eccentricity;
   ConvexArea1(i) = stats.ConvexArea;
end
%% Extract 2's Features
%Initialize vectors to zero
circularity2 = zeros(num2s, 1);
perimeter2 = zeros(num2s, 1);
%Use matlab's built in function to extract features
for i = 1:num2s
   stats = regionprops(bw2seperated(:,:,i), 'Circularity', 'Perimeter');
   circularity2(i) = stats.Circularity;
   perimeter2(i) = stats.Perimeter;
end
%% Extract 0's Features from test data
%Initialize vectors to zero
circularity0_test = zeros(num0s_test, 1);
eccentricity0_test = zeros(num0s_test, 1);
ConvexArea0_test = zeros(num0s_test, 1);
perimeter0_test = zeros(num0s_test, 1);
%Use matlab's built in function to extract features
for i = 1:num0s_test
   stats = regionprops(bw0seperated_test(:,:,i), 'Circularity', ...
                       'Eccentricity', 'ConvexArea', 'Perimeter');
   circularity0_test(i) = stats.Circularity;
   eccentricity0_test(i) = stats.Eccentricity;
   ConvexArea0_test(i) = stats.ConvexArea;
   perimeter0_test(i) = stats.Perimeter;
end
%% Extract 1's Feature from test data
```

```
%Initialize vectors to zero
eccentricity1_test = zeros(num1s_test, 1);
ConvexArea1_test = zeros(num1s_test, 1);
%Use matlab's built in function to extract features
for i = 1:num1s_test
   stats = regionprops(bw1seperated_test(:,:,i),'Eccentricity', 'ConvexArea');
   eccentricity1_test(i) = stats.Eccentricity;
   ConvexArea1_test(i) = stats.ConvexArea;
end
%% Extract 2's Features from test data
%Initialize vectors to zero
circularity2_test = zeros(num2s_test, 1);
perimeter2_test = zeros(num2s_test, 1);
%Use matlab's built in function to extract features
for i = 1:num2s_test
   stats = regionprops(bw2seperated_test(:,:,i), 'Circularity', 'Perimeter');
   circularity2_test(i) = stats.Circularity;
   perimeter2_test(i) = stats.Perimeter;
end
%% Create Feature matrices for all pixel values
featureMatrixAllPixels0 = zeros(num0s, 28 * 28);
featureMatrixAllPixels0test = zeros(num0s_test, 28 * 28);
featureMatrixAllPixels1 = zeros(num1s, 28 * 28);
featureMatrixAllPixels1test = zeros(num1s_test, 28 * 28);
featureMatrixAllPixels2 = zeros(num2s, 28 * 28);
featureMatrixAllPixels2test = zeros(num2s_test, 28 * 28);
%Convert 28x28 matrices into 784x1 vectors and add them as a row in the
%feature matrix
for i=1:num0s
   pixelData = bw0seperated(:,:,i);
   featureMatrixAllPixels0(i,:) = pixelData(:);
end
for i=1:num0s_test
   pixelData = bw0seperated_test(:,:,i);
   featureMatrixAllPixels0test(i,:) = pixelData(:);
end
for i=1:num1s
   pixelData = bw1seperated(:,:,i);
   featureMatrixAllPixels1(i,:) = pixelData(:);
end
for i=1:num1s_test
   pixelData = bw1seperated_test(:,:,i);
   featureMatrixAllPixels1test(i,:) = pixelData(:);
end
for i=1:num2s
   pixelData = bw2seperated(:,:,i);
   featureMatrixAllPixels2(i,:) = pixelData(:);
end
for i=1:num2s_test
   pixelData = bw2seperated_test(:,:,i);
   featureMatrixAllPixels2test(i,:) = pixelData(:);
end
%concatenate needed feature matrices to create test and training data
featureMatrixAllPixels01 = [featureMatrixAllPixels0; featureMatrixAllPixels1];
```

```
featureMatrixAllPixels02 = [featureMatrixAllPixels0; featureMatrixAllPixels2];
featureMatrixAllPixels01test = [featureMatrixAllPixels0test;
                                featureMatrixAllPixels1test];
featureMatrixAllPixels02test = [featureMatrixAllPixels0test;
                                featureMatrixAllPixels2test];
%Release unused matrices
clear featureMatrixAllPixels0 featureMatrixAllPixels0test;
clear featureMatrixAllPixels1 featureMatrixAllPixels1test;
clear featureMatrixAllPixels2 featureMatrixAllPixels2test;
%% Clear non feature data
clear bw0 bw0seperated bw0_test bw0seperated_test;
clear bw1 bw1seperated bw1_test bw1seperated_test;
clear bw2 bw2seperated bw2_test bw2seperated_test;
clear ims0 ims0_test ims1 ims1_test ims2 ims2_test;
%% Construct feature Matrices for 0, 1, and 2
%Construct a matrix of augmented zero vectors
zeroMatrix01 = [ConvexArea0, eccentricity0];
%Construct a matrix of augmented one vectors
oneMatrix01 = [ConvexArea1, eccentricity1];
%Construct a matrix of augmented zero vectors
zeroMatrix01_test = [ConvexArea0_test, eccentricity0_test];
%Construct a matrix of augmented one vectors
oneMatrix01_test = [ConvexArea1_test, eccentricity1_test];
%Build Into actual feature and test matrices
featureMatrix01 = [zeroMatrix01; oneMatrix01];
labelMatrix01 = [zeros(num0s,1); ones(num1s, 1)];
featureMatrix01_test = [zeroMatrix01_test; oneMatrix01_test];
labelMatrix01_test = [zeros(num0s_test,1); ones(num1s_test, 1)];
%Construct a matrix of augmented zero vectors
zeroMatrix02 = [circularity0, perimeter0];
%Construct a matrix of augmented one vectors
twoMatrix02 = [circularity2, perimeter2];
%Construct a matrix of augmented zero vectors
zeroMatrix02_test = [circularity0_test, perimeter0_test];
%Construct a matrix of augmented one vectors
twoMatrix02_test = [circularity2_test, perimeter2_test];
%Build Into actual feature and test matrices
featureMatrix02 = [zeroMatrix02; twoMatrix02];
labelMatrix02 = [zeros(num0s,1); ones(num2s, 1) * 2];
featureMatrix02_test = [zeroMatrix02_test; twoMatrix02_test];
labelMatrix02_test = [zeros(num0s_test,1); ones(num2s_test, 1) * 2];
%Clear non array data
clear zeroMatrix01 zeroMatrix01_test oneMatrix01 oneMatrix01_test;
clear zeroMatrix02 zeroMatrix02_test twoMatrix02 twoMatrix02_test;
clear circularity0 circularity0_test circularity2 circularity2_test;
clear ConvexArea0 ConvexArea0_test ConvexArea1 ConvexArea1_test;
clear eccentricity0 eccentricity0_test eccentricity1 eccentricity1_test;
clear perimeter0 perimeter0_test perimeter2 perimeter2_test;
%% Find k nearest neighbors and record time/accuracy
%Using two features for each feature vector
%Run classifier for 0s and 1s with both city block and euclidean distances,
%as well as k values of 1, 3, 5, and 11
classifierTypes = {"City Block ", "Euclidian "};
for i=[1 2]
```

```
   for j=[1 3 5 11]
       fprintf("-----------------------------------------------------\n");
       string = strcat("Zeros and Ones K Nearest Neighbor Classifier with\nk = %d
using ", ...
                       classifierTypes{i}, "Distance Measurement " + ...
                       "and\nTwo Features\n");
       fprintf(string , j);
       tic
       outputLabels = kNearestNeightbor(featureMatrix01, labelMatrix01,
featureMatrix01_test, j, i);
       toc

       error = CalculateClassificationError(labelMatrix01_test, outputLabels);
       fprintf("Error: %2.2f %%\n", error * 100);
       fprintf("-----------------------------------------------------\n\n");
   end
end
%Run classifier for 0s and 2s with both city block and euclidean distances,
%as well as k values of 1, 3, 5, and 11
for i=[1 2]
   for j=[1 3 5 11]
       fprintf("-----------------------------------------------------\n");
       string = strcat("Zeros and Two K Nearest Neighbor Classifier with\nk = %d
using ", ...
                       classifierTypes{i}, "Distance Measurement " + ...
                       "and\nTwo Features\n");
       fprintf(string , j);
       tic
       outputLabels = kNearestNeightbor(featureMatrix02, labelMatrix02,
featureMatrix02_test, j, i);
       toc

       error = CalculateClassificationError(labelMatrix02_test, outputLabels);
       fprintf("Error: %2.2f %%\n", error * 100);
       fprintf("-----------------------------------------------------\n\n");
   end
end
%% All 28 x 28 pixels
%Using raw pixel data as a 784x1 feature vector
%Run classifier for 0s and 1s with both city block and euclidean distances,
%as well as k values of 1, 3, 5, and 11
for i=[1 2]
   for j=[1 3 5 11]
       fprintf("-----------------------------------------------------\n");
       string = strcat("Zeros and Ones K Nearest Neighbor Classifier with\nk = %d
using ", ...
                       classifierTypes{i}, "Distance Measurement " + ...
                       "and\nAll Pixel Data\n");
       fprintf(string , j);
       tic
       outputLabels = kNearestNeightbor(featureMatrixAllPixels01, labelMatrix01,
featureMatrixAllPixels01test, j, i);
       toc
```

```
        error = CalculateClassificationError(labelMatrix01_test, outputLabels);
        fprintf("Error: %2.2f %%\n", error * 100);
        fprintf("-----------------------------------------------------\n\n");
    end
end
%Run classifier for 0s and 2s with both city block and euclidean distances,
%as well as k values of 1, 3, 5, and 11
for i=[1 2]
    for j=[1 3 5 11]
        fprintf("-----------------------------------------------------\n");
        string = strcat("Zeros and Twos K Nearest Neighbor Classifier with\nk = %d
using ", ...
                        classifierTypes{i}, "Distance Measurement " + ...
                        "and\nAll Pixel Data\n");
        fprintf(string , j);
        tic
        outputLabels = kNearestNeightbor(featureMatrixAllPixels02, labelMatrix02,
featureMatrixAllPixels02test, j, i);
        toc

        error = CalculateClassificationError(labelMatrix02_test, outputLabels);
        fprintf("Error: %2.2f %%\n", error * 100);
        fprintf("-----------------------------------------------------\n\n");
    end
end
```

K Nearest Neighbor Function

```
function labelVector = kNearestNeightbor(trainingData, trainingLabels, testVectors,
k, p)
   %Initialize a blank vector of classifications to output
   labelVector = zeros(length(testVectors(:,1)), 1);
   %For every test vector
   for i=1:length(testVectors(:,1))
       %Initialize a k long list of closest distances and their labels
       distances = Inf(k, 1);
       labels = zeros(k, 1);
       %For all training data
       for j=1:length(trainingData(:,1))
           %Calculate the distance between training vector and test vector
           tempDistance = MinkowskiDistance(trainingData(j, :)', ...
                                            testVectors(i, :)', p);
           %If distance is less than any distances on the closest k
           %vectors find out what position it would fall on the list
           distanceIndicies = tempDistance < distances;
           distanceIndex = find(distanceIndicies, 1, 'first');
           %Insert the distance and label in to the list of closet and
           %pop the new farthest vector
           if ~isempty(distanceIndex)
               distances = [distances(1:(distanceIndex - 1))' tempDistance
distances(distanceIndex:end)'];
               labels = [labels(1:(distanceIndex - 1))' trainingLabels(j,1)
labels(distanceIndex:end)'];
               distances(k + 1, :) = [];
               labels(k + 1, :) = [];
           end
       end
       %After going through all training vectors assign the label to the
       %test vector as the most frequent label on the k closet vectors
       %label list
       labelVector(i, 1) = mode(labels);
   end
end
```

Minkowski Distance Function

```
function distance = MinkowskiDistance(referenceVector,inputVector, p)
   %Compute distance between two vectors based on p, equation taken from
   %Pattern Classification by Richard O. Duda, Peter E. Hart, and
   %David G. Stork. Equation 58 in chapter 4
   distance = (sum((abs(inputVector - referenceVector)) .^ p)) .^ (1 / p);
end
```

Bar Graph Script

```
%% Two Features Error
Error01 = [1.82 1.31 1.26 1.16;
1.82 1.31 1.26 1.16];
Labels = categorical({'City Block','Euclidian'});
barGraph(Error01,Labels,"Error (%)");
%%
Error02 = [9.27 7.27 6.78 6.88;
9.37 7.13 6.98 6.78];
Labels = categorical({'City Block','Euclidian'});
barGraph(Error02,Labels,"Error (%)");
%% Pixel Features Error
Error01 = [0.10 0.15 0.15 0.15;
0.10 0.15 0.15 0.15];
Labels = categorical({'City Block','Euclidian'});
barGraph(Error01,Labels,"Error (%)");
%%
Error02 = [0.90 0.85 0.85 1.00;
0.90 0.85 0.85 1.00];
Labels = categorical({'City Block','Euclidian'});
barGraph(Error02,Labels,"Error (%)");
%% Two Features Time
% 01 & 02 per sample
Time01 = [27.06 27.00 27.59 27.50;
27.33 27.47 27.18 27.64];
Time01=round(Time01,1);
Time02 = [25.49 25.17 25.44 25.82;
25.44 25.33 25.44 25.83];
Time02=round(Time02,1);
Labels = categorical({'01 City Block','01 Euclidian','02 City Block','02
Euclidian'});
barGraph([Time01;Time02],Labels,"Time (ms)");
%%
% 01 & 02 Total
Time01 = [53625 53516 53677 54507;
54169 54437 53867 54781];
Time01=round(Time01./1000,1);
Time02 = [51155 50517 51065 51829;
51059 50846 51061 51845];
Time02=round(Time02./1000,1);
Labels = categorical({'01 City Block','01 Euclidian','02 City Block','02
Euclidian'});
barGraph([Time01;Time02],Labels,"Time (s)");
%% Pixel Features Time
% 01 & 02 per sample
Time01 = [139.6 137.8 132.7 131.9;
129.6 127.7 128.8 129.1];
Time01=round(Time01,1);
Time02 = [122.0 124.0 124.2 124.7;
118.7 120.6 123.4 121.4];
Time02=round(Time02,1);
Labels = categorical({'01 City Block','01 Euclidian','02 City Block','02
```

```
Euclidian'});
barGraph([Time01;Time02],Labels,"Time (ms)");
%%
% 01 & 02 Total
Time01 = [276743 273491 263203 261453;
256952 253322 255192 255928];
Time01=round(Time01./1000,1);
Time02 = [244922 248798 249247 250264;
238301 242101 247680 243651];
Time02=round(Time02./1000,1);
Labels = categorical({'01 City Block','01 Euclidian','02 City Block','02
Euclidian'});
barGraph([Time01;Time02],Labels,"Time (s)");
```

Bar Graph Function

```
function barGraph(graphData,x_labels,y_label)
%barGraph Wrapper function for bar graphs for project 6
%xlabels = categorical({'01 City Block','01 Euclidian','02 City Block','02
Euclidian'});
b = bar(x_labels,graphData);
ylabel(y_label);
legend("1","3","5","11");
% data at top of bars
for i = 1:length(b);
xtips1 = b(i).XEndPoints;
ytips1 = b(i).YEndPoints;
labels1 = string(b(i).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
'VerticalAlignment','bottom')
end
end
```