ECE 662
Statistical Pattern Recognition and Decision Making Processes

Homework #3

Non-Parametric Density Estimation

29 April 2010

Hungry H. Hippo

Abstract

In this paper, non-parametric density estimation is explored through the use of two classification techniques: the Parzen Window decision method, and the K-Nearest Neighbor (KNN) decision method. In each of the two experiments performed, multiple tests were performed. All data was synthetically generated, Normally distributed, and independent. In the Parzen Window experiment, the size of a cubic window is varied in one test, and the number of training samples used in the decision is varied the second. For the KNN Experiment, initially the number of training samples used in the decision is varied, and the number of neighbors used (K) is varied in the second test.

Experiment One: The Parzen Window Decision Method

The Parzen window method places a "window" around a sample point to be classified, and collects the training data points from every class falling within this window, and weights them accordingly. In the case of a cubic window, the following weights are used:

$$\text{weight}(\overrightarrow{X_i}) = \begin{cases} 1, & X_{i,k} \in [X_0 - \frac{WinSide}{2}, \quad X_0 + \frac{WinSide}{2}] \\ 0, & otherwise \end{cases}$$

The final decision goes to the class which has the highest total weight in the window surrounding X_0.

In this experiment, a cubic Parzen window is used in every case. In the first test, the error rate relative to the window width is explored, with runs in several different numbers of features. In test #2, the error rate relative to the Number of Training samples used on the data is explored, again with runs in several different dimensional spaces.

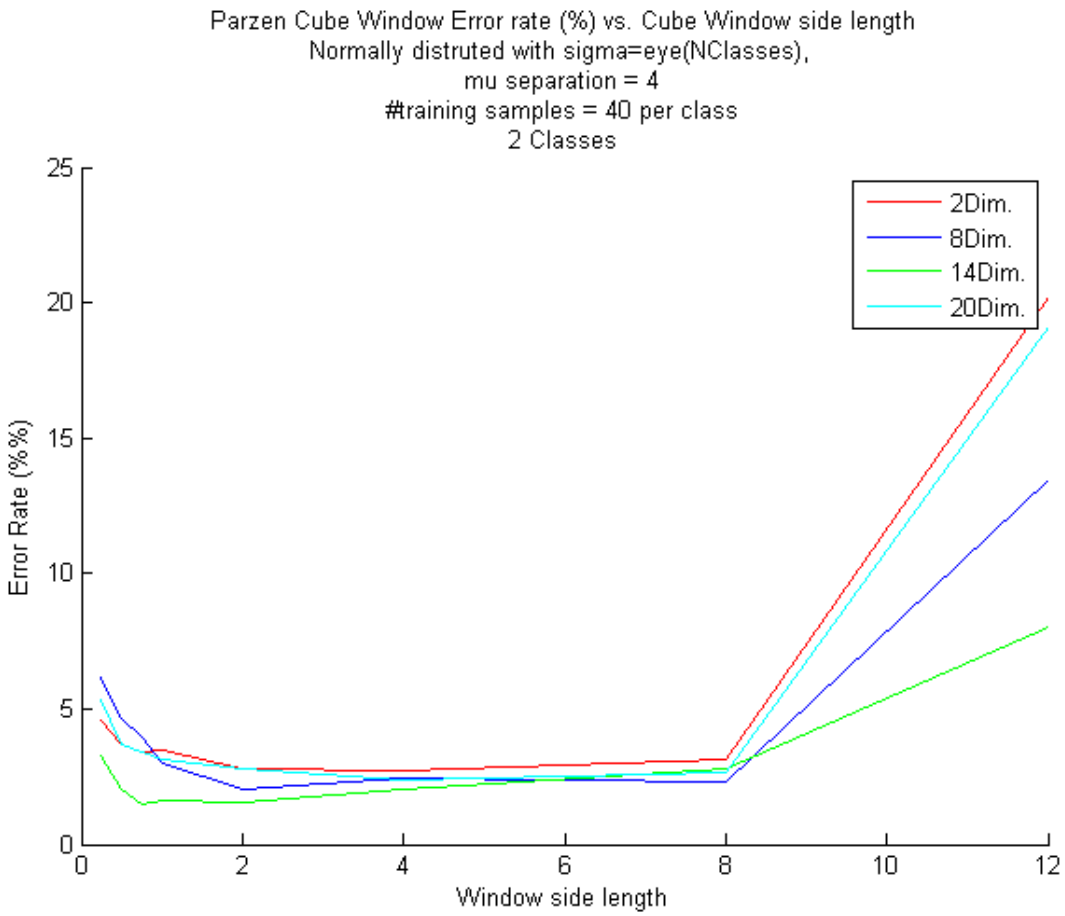

Figure 1 Experiment One, Test #1

Test 1: Vary Parzen Window Size

Test Setup

For this test, the error rate of this Parzen Cubic window classifier was determined for several different window edge sizes, taking on values of [.25, .5, .75, 1, 2, 4, 8, 12]. For each classifier problem, 40 training samples were used for the classifier for 2 classes, both Normally distributed. Each variable is independent with equal scatter. For reference, the standard deviation is one, equal to the window size at one point on the graph.

This basic experiment was repeated several times in a feature space with increasing dimension, taking on values of [2, 8, 14, 20]. In addition, one extra run in 2 dimensions, with 7 classes was run.

Results: Run #1

The basic idea for this test was to show that a window which is too small or too large will degrade the accuracy of the classifier. This premise held very well. From Figure 1, one can see that the undersized Parzen window results in a roughly 2 percent increase in classification errors, or about 4% in the worst case (8 dimensions). Similarly, an oversized Parzen window quickly destroys the accuracy of this classifier, adding upwards of 15% more failures in some cases. This makes sense, since the window

more or less consumes both distributions at this point! The mean separation for this test was 4 in each dimension. A window which was 12 units on a side would surely take up most of each class.

In the end, adding more dimensions to the feature space (i.e. adding more independent variables) to the data set did not reliably reduce the error rate. The 14 dimensional case performed better than the 20 dimensional case. Note that the data was randomly generated, And has not been smoothed.

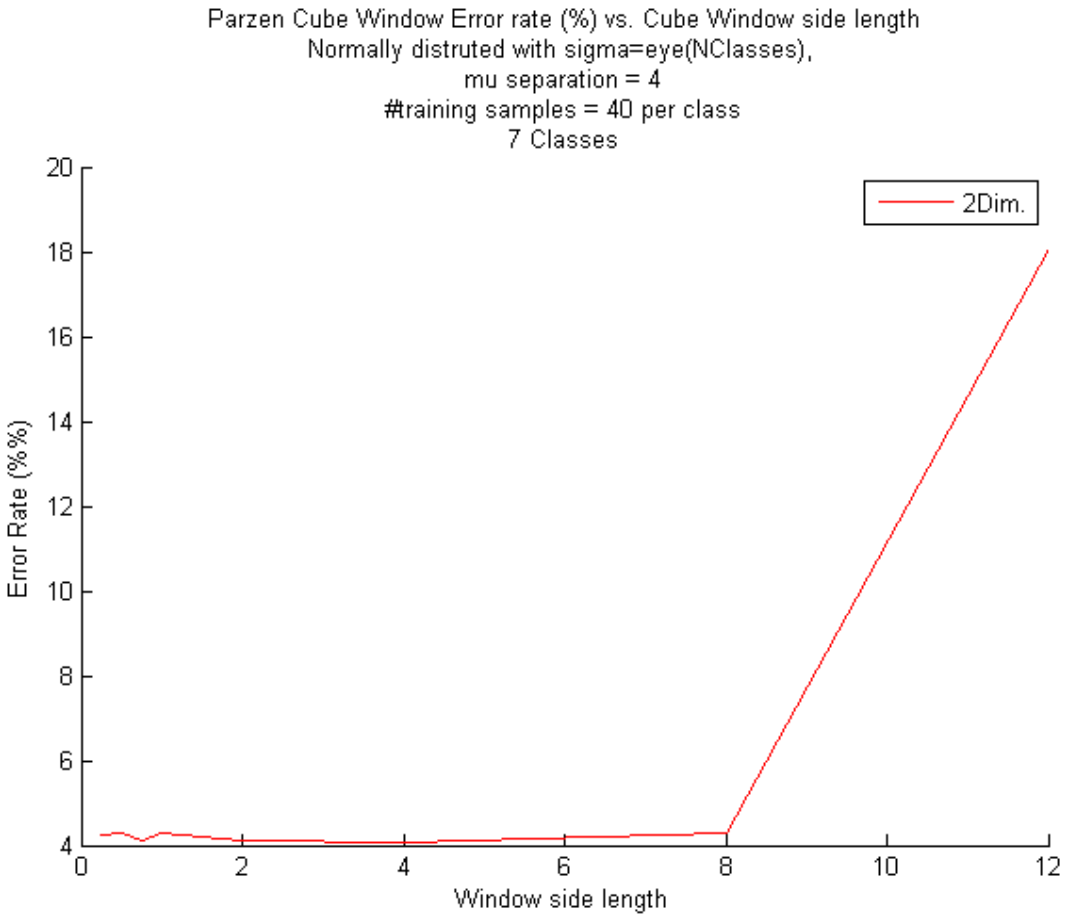

Figure 2 Experiment 1 Test 1, run #2

Results: Run #2

In Figure 2, one can see the same test performed when 7 classes are present, all Normally distributed in 2 dimensions. The classes are arranged as in Figure 3, such that some classes of two potential sources of error. (Arranging them equidistant in N-space is trickier than it would seem at first.) During the test, the entire run took nearly an order of magnitude longer to execute, although it used 3.5 more training samples in each case. As a result, only the 2-dimensional case was completed.

From Figure 2, one can see the error rate reaches a minimum of about 4%. The undersized window case only contributes less than 1% to the error rate, and does not have as dramatic of an effect

as the 2-class case from Run #1. However, the oversized Parzen window follows a similar pattern of error as the 2-class case in 2 dimensions.

Essentially, adding more classes does not improve the accuracy of the classifier, but also did not degrade it by more than about 2%.

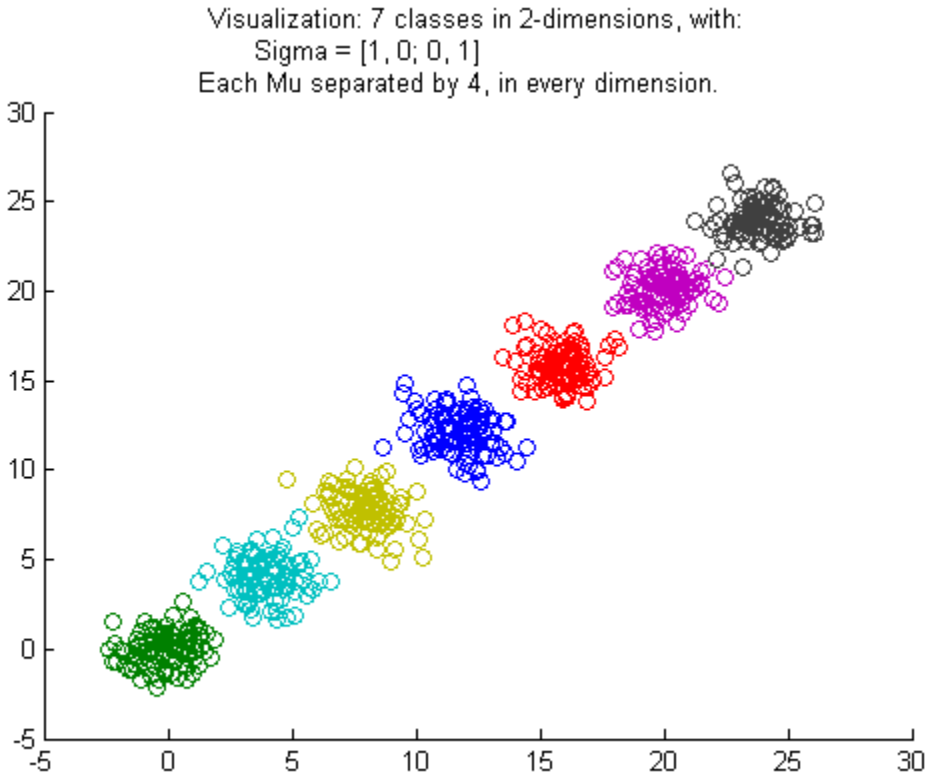

Figure 3 Experiment 1, Test 1, Run #2

Test 2: Vary Number of training samples used

Test Setup

For this test, the error rate of this Parzen Cubic window classifier was determined for several different numbers of training samples, using one of [5 10 20 50 75 100 200] samples for each *class*, so twice that in the 2 class case. The window size was fixed at 1, and each variable is independent Normal with equal scatter of 1.

This basic experiment was repeated several times in a feature space with increasing dimension, taking on values of [2, 8, 14, 20].

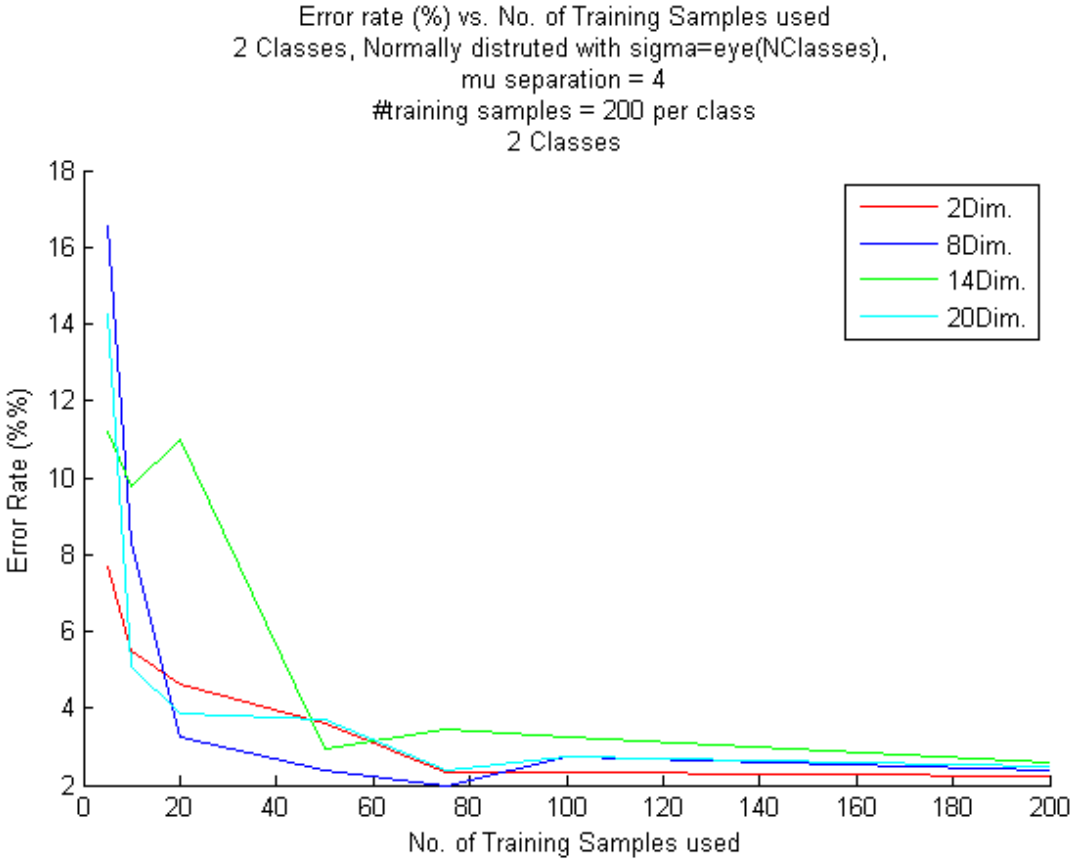

Figure 4, Experiment 1, Test 2 Results

Results:

This test was designed to test the effect of an under-trained Parzen Window classifier. The results are shown in Figure 4. From the figure, the classifier's error rate suffers severely for less than about 20 training samples per class for this data set. At that number, the space is relatively sparse, and the window does not capture enough training samples to be effective.

After 75 samples, the error rate continues to decline for every case. Notice that in Test #1, we had seen that the N=14 dimension case performed best for an undersized window, but in the under-trained case it performs the worst. Further tests could show a possible correlation there.

Experiment Two: The K Nearest-Neighbor Decision Method

For the K Nearest Neighbor (KNN) decision method, several distance measures are considered. Each measure is a derivative of the p-norm, i.e.

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

To make a decision, the K-nearest training sample points are selected, and the class with the greatest number of samples is selected. In the event of a tie, this experiment chooses to blindly guess.

In the first KNN test, the error rate relative to the number of training samples is explored, with several runs using the following norm functions: [L1-Norm, L2-Norm, L_{max}-Norm]. In the second test, the error rate relative to K (the number of neighbors used) is varied while the error rate is recorded. Again, this test runs using three norm functions.

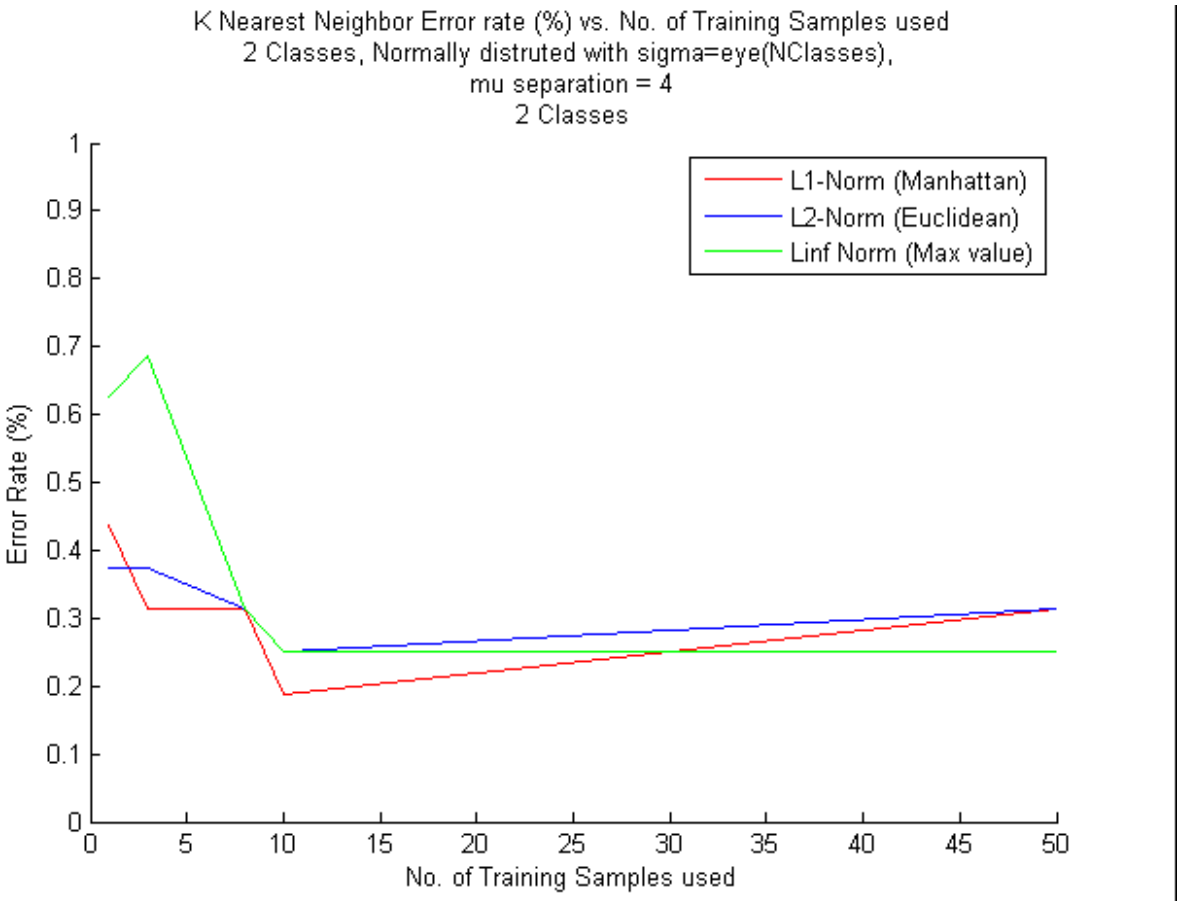

Figure 5, Experiment 2, Test 1 Results

Test 1: Vary the Number of training samples used

Test Setup:

This test used 2 independent Normal classes with mu separation of 4, and unit variance. To test the robustness of the KNN classifier, the number of samples used in the classifier is varied from 3 to 50. For each basic test, it was performed using three different distance functions: the L_1 norm (known as the "taxicab norm"), the L_2 norm (or Euclidean norm), and the L_{max} norm (which essentially uses the largest value for the result). In this test, K=1 nearest point.

Results:

As expected, the accuracy of the classifier decreases when too few samples are used. However, it is important to note that the accuracy of this classifier is still extremely high; when using 3 samples per class, a total of 6 points, a success rate of 99.3% is achieved! Increasing the number of training points gives at most 0.4% improvement for this data set. Overall, one can expect the KNN method to work relatively well when very few training samples are available.

Notice that this test is equivalent to the "Nearest Neighbor rule," which ran extremely quickly when compared to the Parzen Window method in Experiment 1. In comparison, this method showed a lower error rate as well.

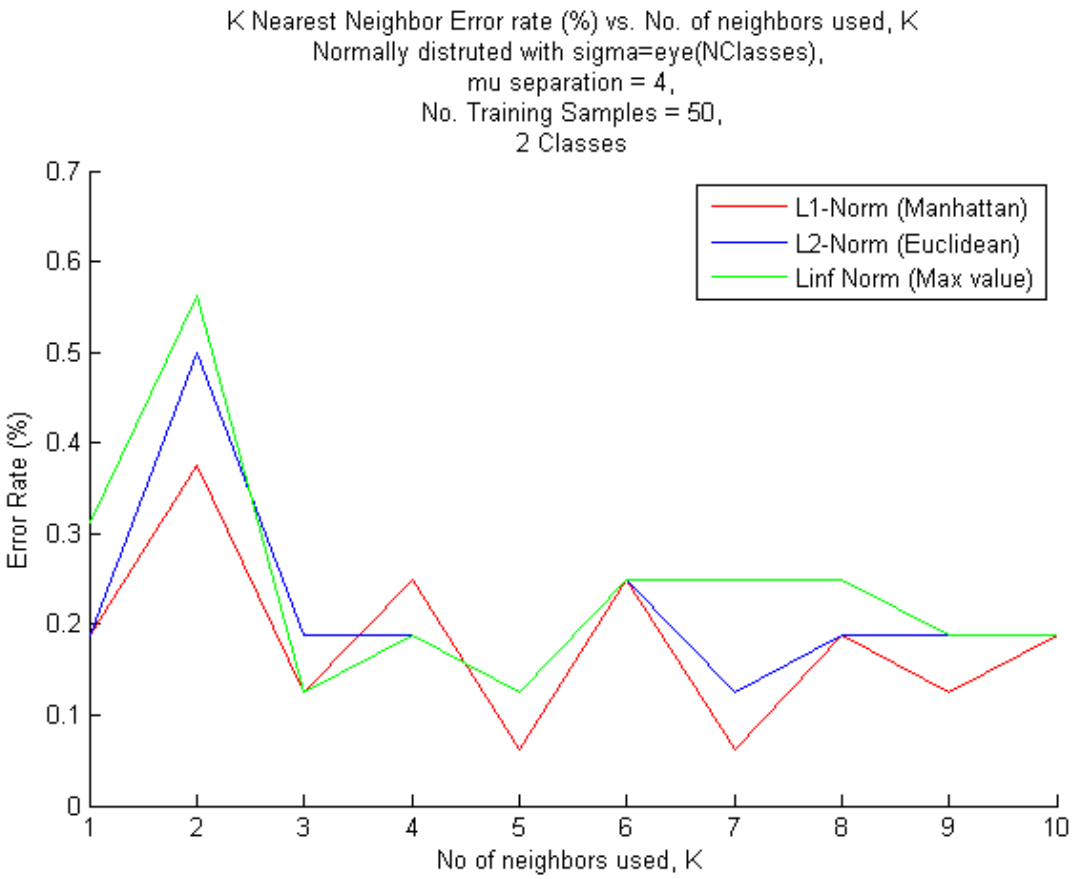

Figure 6, Experiment 2, Test 2, Run #1

Test 2: Vary the Number of Neighbors (K)

Test Setup:

Similar to test #1 in Experiment 2, this test uses 2 independent Normal classes with a mu separation of 4, and unit variance. To test the most critical parameter of this method, the number of neighbors used in the decision, K, is varied from 1 to 10. Each class uses 50 training samples.

This test is repeated for the 7 class case as well, with all other variables equal.

Results, Run #1:

Figure 6 shows the results of the first run of this test. In the 2-class case, this classifier performed very well, consistently achieving over 99% accuracy. To get the most out of a KNN classifier, however, the choice of K can reduce the error rate by as much as 0.3% for this data set, when the Euclidean norm is used.

One can see a "zig-zag" pattern appear from the figure. One can see that every even value chosen for K, for 2 classes, results in a higher failure rate. This gives concrete evidence to the widely used suggestion to always choose an odd K for 2-class classifiers.

Overall, however, using K=1 will give very good results without much thought, and an odd K from 3 to 7 can yield slightly better performance.

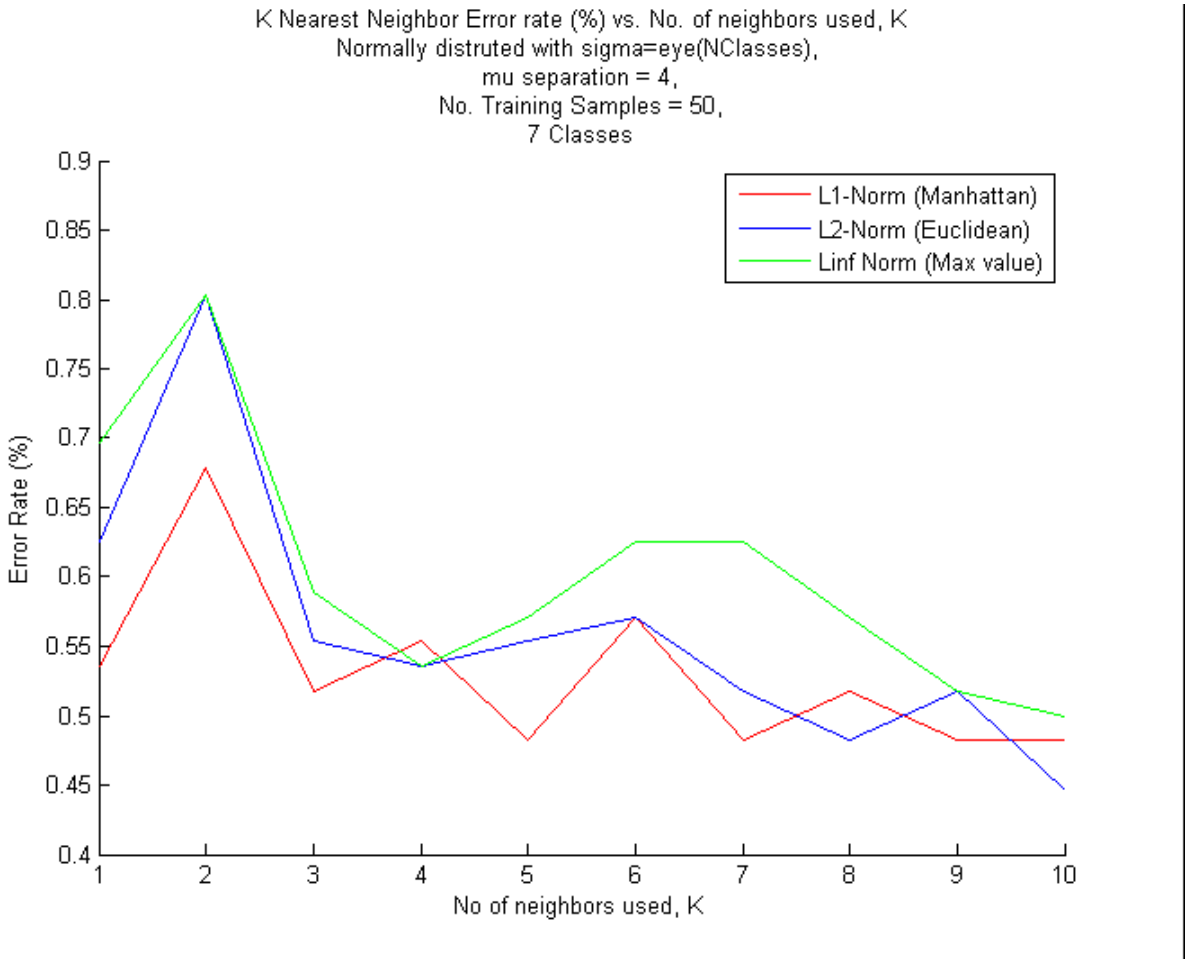

Figure 7, Experiment 2, Test #2, Run #2

Results, Run #2:

In this test run, 7 classes were present, instead of only 2. Each class is positioned according to Figure 2.

From Figure 7, one can see that the error rate continues to fall as it approaches K=10, so more neighbors may be needed for an increased number of classes. In this data set, the classes are linearly separated, so it is not as much of an advantage. However, for hypothetical classifier which may see several classes represented in the K nearest training points, it may be worthwhile to test for various K values above the number of classes.

As in Run #1 of this test, the choice of K can affect the precision of a KNN classifier by < 1%, but overall the classifier performs very well, with over 99% success. However, the suggestion to choose an odd K is unfounded for the Eudidean distance function, although the Manhattan distance strangely still appears to perform better for odd K values. This may be due to the layout of the means, however.

Conclusions

In summary, the behavior of two non-parametric density estimators was explored by varying the number of training samples used, as well as a fundamental parameter value (the window size for the Parzen method, or K for the KNN method). At the most basic level, the K Nearest-Neighbor decision method worked for roughly 99% of cases, while the Parzen Window method only achieved roughly 97% accuracy. The Parzen method was more sensitive to its setup parameters (Window size and number of training samples) but still performed very well. The KNN method was relatively robust to fluctuations in training data or selection of K, and proved to be the simplest to implement.

In future test, it would be extremely beneficial to use a Gaussian Window for the Parzen method, as this is a more widely accepted method. A different window selection may prove to beat the KNN success rate, but in practice it was prone to a longer development time.

For more separable data, one can be confident either method would work effectively. However, for more “creatively” distributed data sets, further experiments would be required. That, or one could take the easy route and use the KNN method.