ECE 662

Homework 2

April 1, 2008

Question 1: In the Parametric Method section of the course, we learned how to draw a separation hyperplane between two classes by obtaining w_0, the argmax of the cost function $J(w)=w^T S_B w / w^T S_w w$. The solution was found to be $w_0= S_w^{-1}(m_1-m_2)$, where m_1 and m_2 are the sample means of each class, respectively.

Some students raised the question: can one simply use $J(w)= w^T S_B w$ instead (i.e. setting S_w as the identity matrix in the solution w_o? Investigate this question by numerical experimentation.

In order to compare the results of changing the cost function to $J(w)= w^T S_B w$ from $J(w)=w^T S_B w / w^T S_w w$, the within class scatter matrix S_w is set to the identity matrix. In Matlab, two classes are created with two feature vectors of normal distribution. Equal numbers of data points are taken from each class. The mean values for class one is set to value [3, 3] and the mean for class two is set to [5, 5]. The one covariance matrix is set as the identity matrix while the second is varied across the main diagonal with values from 0.1 to 2 in increments of 0.1. For each case 300 data points are created using $w_0= S_w^{-1}(m_1-m_2)$ for one case and $w_0= (m_1-m_2)$ for the second. Lastly 5,000 samples are created and the ratio of errors for the two methods is calculated for each of the twenty variances.

The ratio of errors is shown in figure 1.1. The minimum ratio and maximum ratio are 0.5690 and 1.0847 respectively. This is somewhat surprising to see that the simplified version of the cost function is sometimes better than the original.

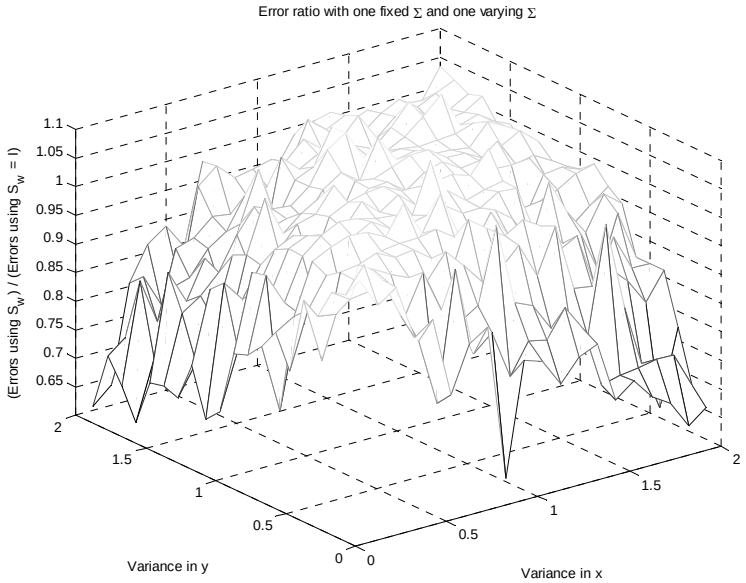


Figure 1.1 Error ratio of cost function using S_w / cost function not using S_w with changing variances

One would expect that the results from the original cost function which takes the within class scatter into account is not always better than (in two dimensions) taking the perpendicular bisector of the means of classes. Figure 1.2 shows an example where not taking within class scatter would cause more errors in classification than without taking the within class scatter into account.

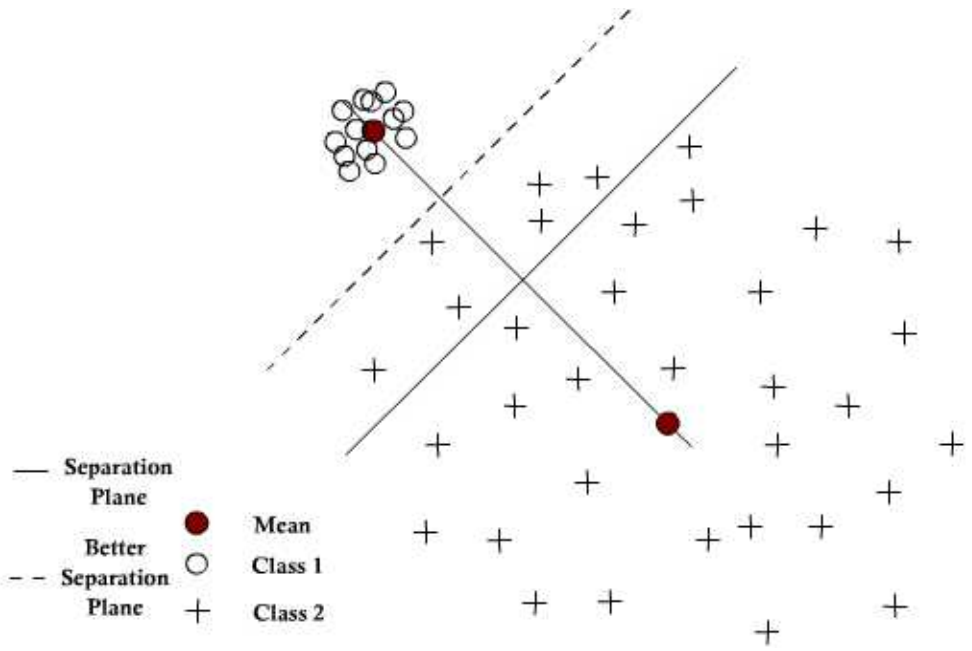
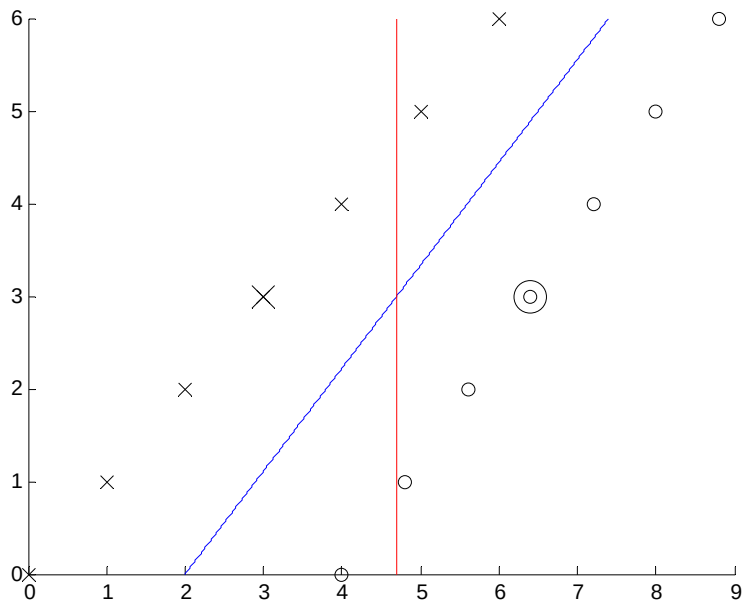


Figure 1.2 Example of problematic spread in 2-D

Another example is displayed in figure 1.3, where the large X and the two concentric O circles are the means of the two classes. The perpendicular line is the hyperplane created by taking S_w as the identity matrix and the diagonal line is using the original cost function. This shows an example where the within class scatter is important for the decision hyperplane.



In conclusion, the scatter is important for creating a separation hyperplane, but the simplified function can still yield decent results. If there is a constraint on computational complexity, then the simplified version may still be acceptable for fast results, perhaps if needed in real time applications.

Question 2: Obtain a set of training data. Divide the training data into two sets. Use the first set as training data and the second set as test data.

a) Experiment with designing a classifier using the neural network approach.

b) Experiment with designing a classifier using the support vector machine approach.

c) Compare the two approaches.

Note: you may use code downloaded from the web, but if you do so, please be sure to explain what the code does in your report and give the reference.

Neural Networks

Matlab code was used from "ANN: DTU Toolbox" at URL http://isp.imm.dtu.dk/toolbox/ann/. The code uses a variable number of hidden layers, which are each hyperbolic tangent functions. Separate functions allow the output of the code to be probabilities of the two classes. The code optimizes the weights with a maximum a posteriori approach. This data used for this experiment, and following experiments is taken from the website http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ and uses the "svmguide1" dataset. This contains two classes, four features, a training size of 3,089, and testing size of 4,000. To speed up the neural network algorithm, the data set is cropped down to 150 samples from each class and uses all four features. The number of hidden layers is varied from 1 to 20. Figure 2.1 shows the percent error as the number of hidden layers change.

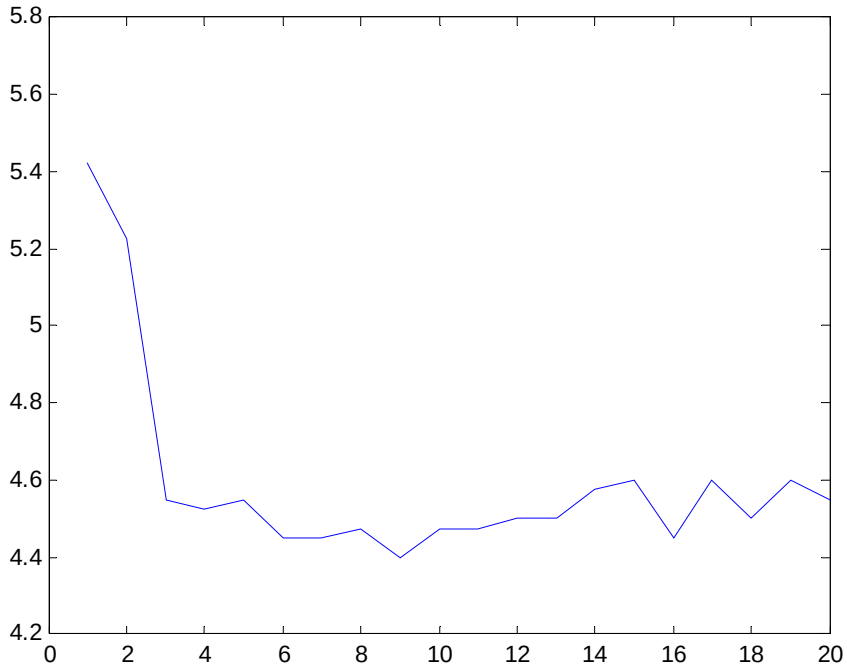


Figure 2.1 Percent error as a function of number of hidden layers.

The minimum percent error is found to be 4.4%, which occurs at nine hidden layers. From the graph, the increasing number of hidden layers does not provide much in terms of lowering the percent error after three hidden layers.

Support Vector Machines

For this experiment, code from Matlab's bioinformatics toolbox is used. The same cropped 150 samples from each class are used from the svmguide1 dataset. Matlab's svmclassify function uses a linear support vector machine classifier to pick a separation plane, find the worst classified sample, move the separation plane to improve the worst classified sample, and iteratively repeat until a convergence appears. Svmclassify automatically classifies the test data. The end result percent error was found to be 5.8%.

Comparison

Both of these classifier approaches showed relatively low percentages of error. The major concern is the computational cost and complexity of the algorithm. While only 150 samples were taken from each of the classes, the neural network approach took a noticeable amount of time longer than the support vector machines.

Question 3: Using the same data as for question 2 (perhaps projected to one or two dimensions for better visualization),

a) Design a classifier using the Parzen window technique.
b) Design a classifier using the K-nearest neighbor technique
c) Design a classifier using the nearest neighbor technique.
d) Compare the three approaches.

In order to test and compare the Parzen window, K-nearest neighbor, and nearest neighbor classifier techniques, data is again taken from the svmguide1 dataset. To simplify the coding, two features are used and the training and testing size is reduced to 1000 samples. Figure 3.1 shows the two dimensional locations of each of the training samples. The first graph in figure 3.1 is the combination of both classes. The second two graphs show the circles of class one and the plus signs of class two respectively. Figure 3.2 shows the same two dimensional locations but for the testing data.

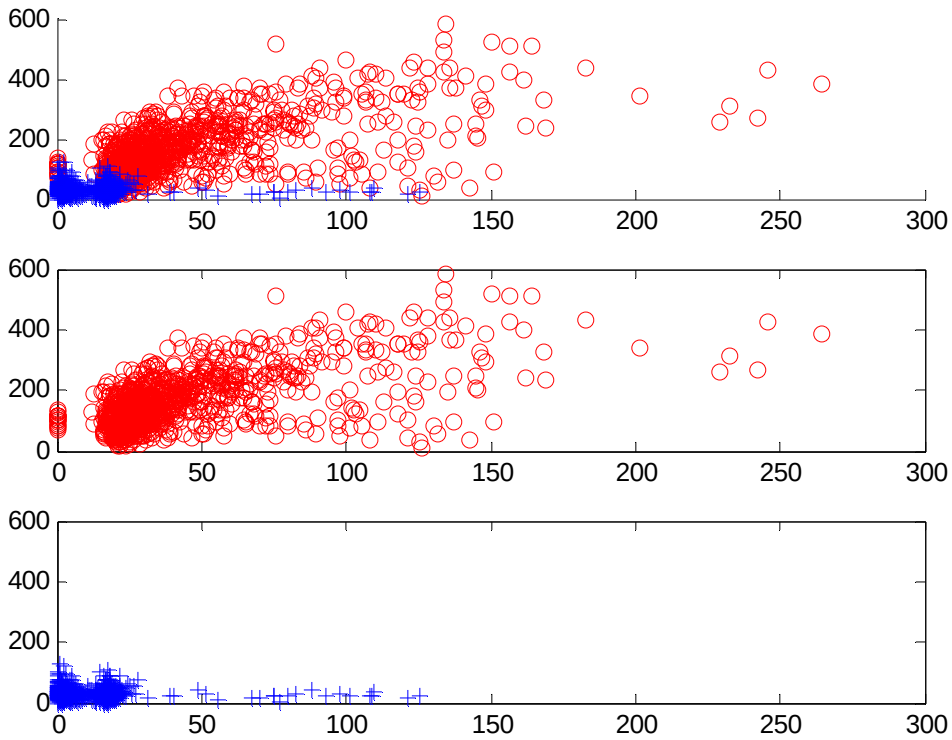


Figure 3.1 Training data

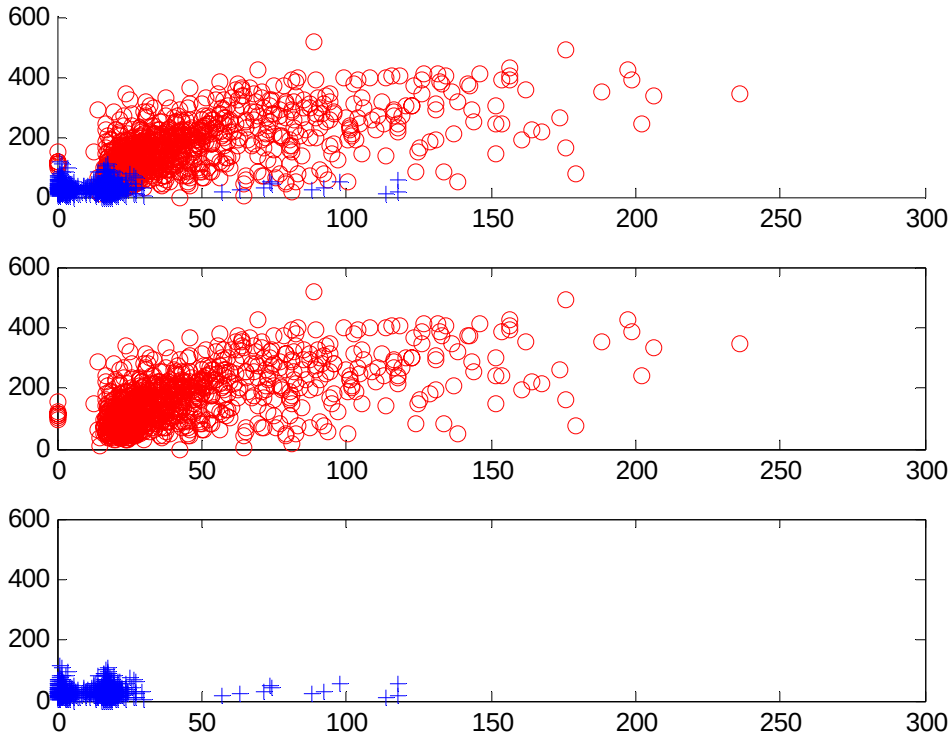


Figure 3.2 Testing Data

Parzen Window

For the Parzen windows, the probability of each class is determined by the number of training points within a circular radius of each test point. The class which has the larger number of training points in the radius classifies the test sample. The radius can be adjusted to see the variation in radius size and percent error.

After running the experiment, a radius of around 20 shows the lowest percent error for this data. The optimal radius is dependent on the training data. If the training data is sparser, then the optimal radius is expected to be higher than 20.

Radius	Percent Error
1	46.55
5	14.75
10	8.90
15	7.90
20	7.55
30	8.00

Table 3.1 Parzen Percent Error vs. Radius

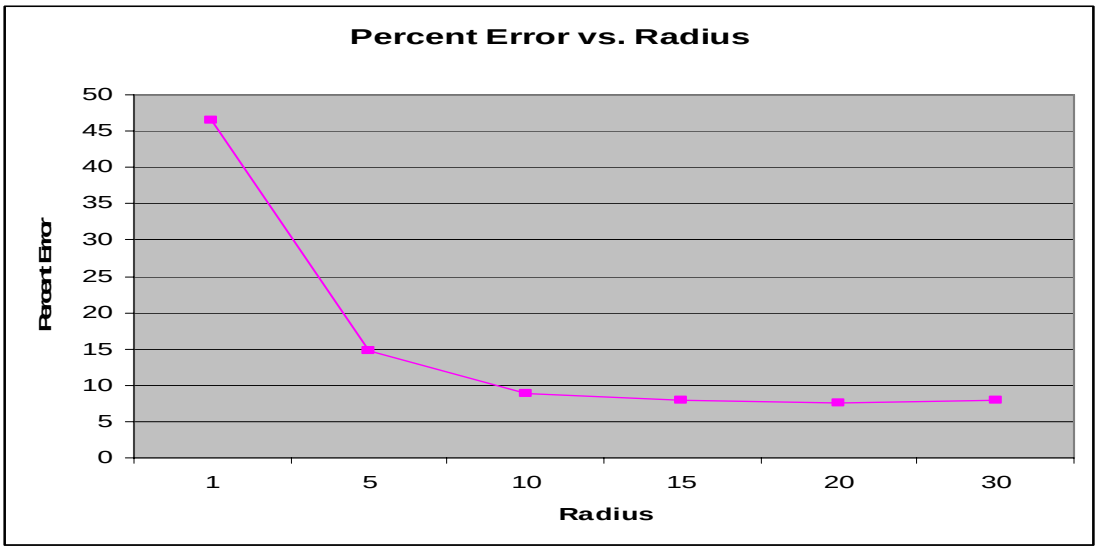


Figure 3.3 Parzen Percent Errors vs. Radius

Looking at the number of training points within the radii of the testing points produces the probability of being in class 1 or class 2. Figure 3.4 shows the number of hits and misses for both classes. The first graph show the number of

class 1 hits for each of the 1000 class 1 test points and the second graph shows the number of class 2 hits (misses) for each test point. When the level of graph 1 is higher than the level of graph 2, then the test point is correctly labeled as class 1. Graphs 3 and 4 shows the hits and misses for the 1000 class 2 test points. Graph 4 shows that in the same radius many hits result, so class 2 has tightly packed sample points. Since class 2 is tightly packed, there are much fewer misses and therefore much fewer misclassifications.

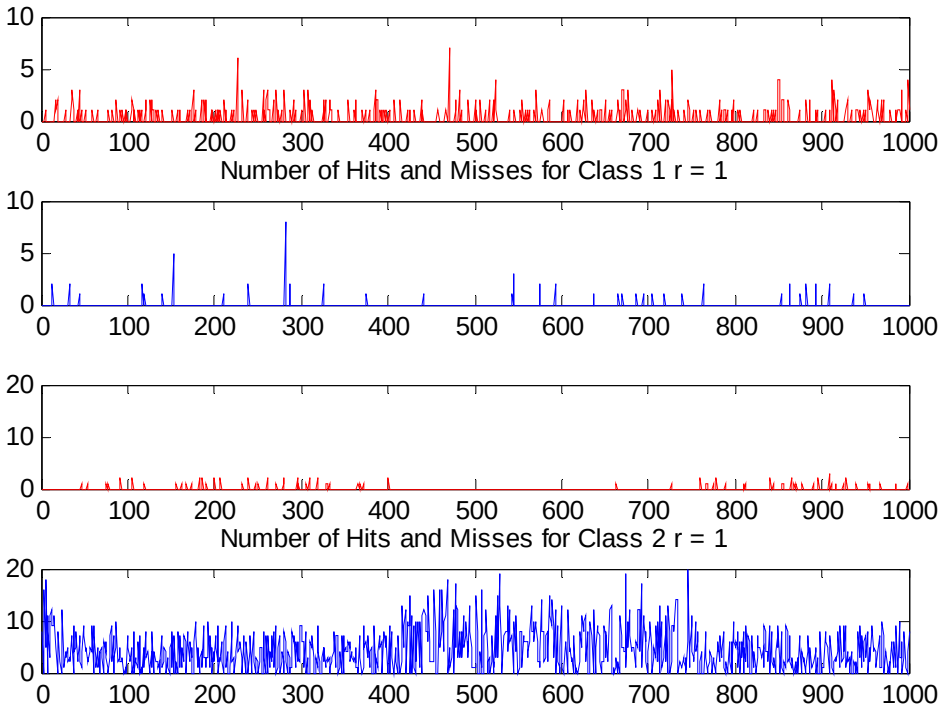


Figure 3.4 Number of hits and misses with radius 1.

K-Nearest Neighbors

For K-nearest neighbors, a value K is chosen to be a natural number. K is typically an odd number and to determine the classification the K nearest neighbors based on a chosen distance parameter are observed. The class with the most training points of those neighbors is the final classification of the sample of interest. In this experiment, Euclidean distance is the distance parameter. The K values are varied.

Using the svmguide1 data points, the K values are varied from 5, 10, and 25. Table 3.2 shows the percent error for each of the K values. For these K values, the percent error did not change very significantly. This is because there is reasonable separation between the two classes.

K value	Percent Error (%)
5	4.6
10	4.6
25	4.8

Table 3.2 K values with percent errors for svmguide1 data.

To better see what is occurring with K-nearest neighbors, a test is run on the applet from http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html. In this applet the number of samples and the K value is varied. Table 3.3 and Figure 3.5 show the results of the experiments.

Number of Samples	Number of Nearest Neighbors (K)	Error Rate (%)
2000	5	3.3750002
2000	10	4.546875
2000	25	6.34375
2000	100	7.953125
1000	5	4.8125
1000	10	5.625
1000	25	7.296875
500	5	6.484375
500	10	7.0468745
500	25	7.8906255

Table 3.3 K nearest neighbor results from website applet.

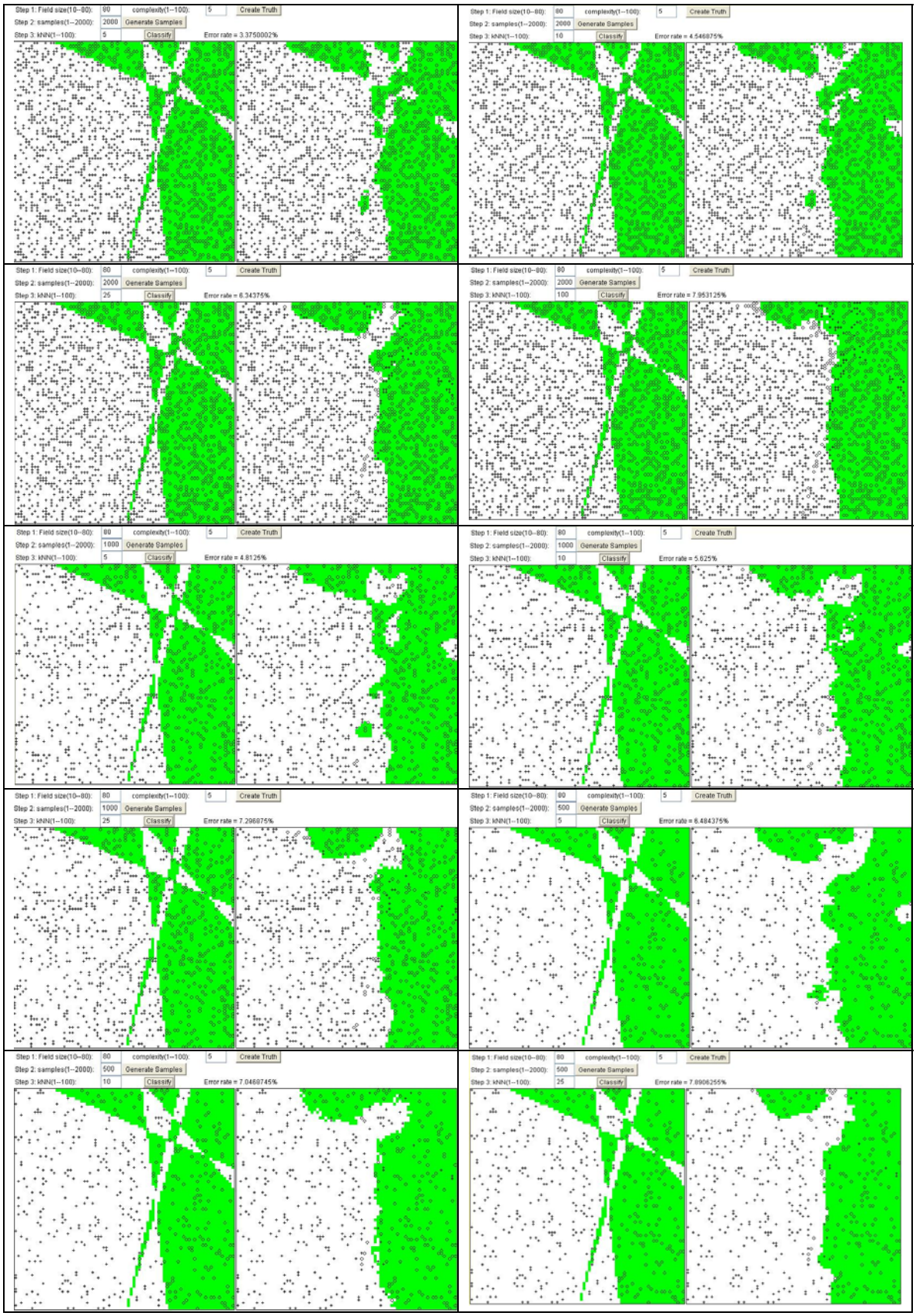

Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 2000 Generate Samples
Step 3: kNN(1~100): 5 Classify Error rate = 3.3750002%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 2000 Generate Samples
Step 3: kNN(1~100): 10 Classify Error rate = 4.546875%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 2000 Generate Samples
Step 3: kNN(1~100): 25 Classify Error rate = 6.34375%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 2000 Generate Samples
Step 3: kNN(1~100): 100 Classify Error rate = 7.953125%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 1000 Generate Samples
Step 3: kNN(1~100): 5 Classify Error rate = 4.8125%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 1000 Generate Samples
Step 3: kNN(1~100): 10 Classify Error rate = 5.625%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 1000 Generate Samples
Step 3: kNN(1~100): 25 Classify Error rate = 7.296875%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 500 Generate Samples
Step 3: kNN(1~100): 5 Classify Error rate = 6.484375%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 500 Generate Samples
Step 3: kNN(1~100): 10 Classify Error rate = 7.0468745%
Step 1: Field size(10~80): 80 complexity(1~100): 5 Create Truth
Step 2: samples(1~2000): 500 Generate Samples
Step 3: kNN(1~100): 25 Classify Error rate = 7.8906255%

Figure 3.5 K nearest neighbors from the website applet.

Looking at the images in figure 3.5, the images in the first and third column are the test points along with the correct classifications. The shaded region is one class and the white region is the other class. The second and fourth columns show the classifications after finding the k nearest neighbors. From these, it can be concluded that the more sample points and smaller K values yield the lowest percent error. The resolution of the output classes is also increased with the more samples and smaller K, although the skinny regions of the classes still cause some problems in the classification. A major assumption with the website applet is that there are no stray points in the training data, so there are no class 1 samples in the class 2 region. This assumption allows the applet to yield lower percent errors than more realistic cases.

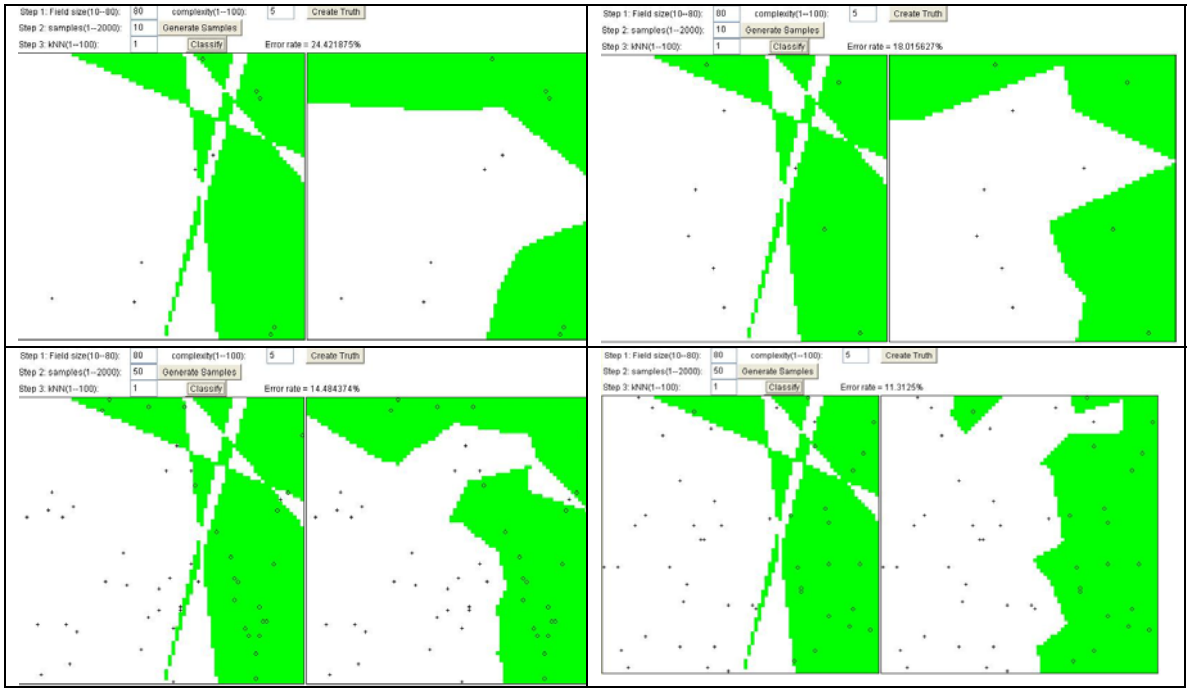
Nearest Neighbors

Nearest neighbors is a simplified case of K-nearest neighbors where K equals 1. In this case, the same Euclidean distance parameter and the K-nearest neighbor code setting K to be equal to 1 are used. The resulting percent error is 6.2%, which is larger than the errors with the K-nearest neighbors.

Using the same website applet but varying the number of samples again shows the more samples the better results. Each experiment is run twice to check the consistency of the error rate. As the sample number increases, the consistency becomes better. Again looking at the resulting graphs, as the number of sample point's increase, the resolution of the classification becomes better. At the 2000 sample case, the thin lines of each class are nearly distinguishable.

Number of Samples	Error Rate (%) 1	Error Rate (%) 2
10	24.421875	18.015627
50	14.484374	11.3125
100	10.78125	8.15625
500	6.28125	5.4062495
1000	4.171875	3.640625
2000	3.078125	3.15625

Table 3.4 Nearest neighbors using the website applet



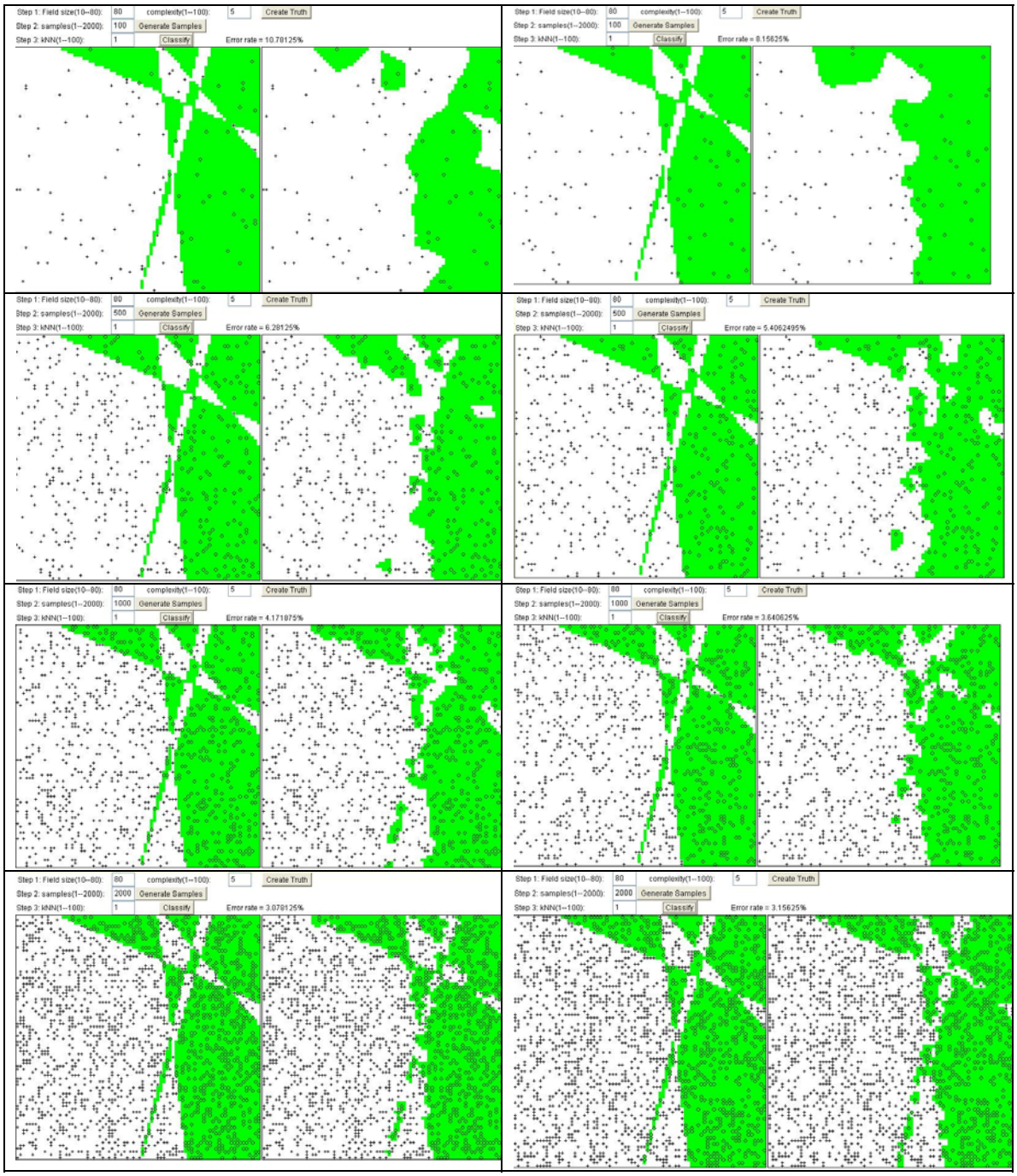


Figure 3.6 Nearest neighbors using the website applet

Comparison

After looking at the three classifications, Parzen windows, K-nearest neighbors, and nearest neighbor, techniques, the K-nearest neighbor's results showed the lowest percent error using real world data. Parzen windows with 7.55%, K-nearest neighbors with 4.2%, and nearest neighbors with 6.2%. There are some concerns with using the blanket statement that the K-nearest neighbors will always yield the best. The classification method is data dependent. So each technique has unique benefits.

As shown in the applet, if there are sharp regions of a single class, then the nearest neighbor will best classify those regions. If there is a good separation between classes, and sample points are not very sparse, then the K-nearest neighbors yield very good results. If samples points are sparse, then Parzen windows may yield the best results. Therefore it is necessary to have prior knowledge of the samples before choosing the classification technique.