

Classification Using Linear Discriminant Analysis and Quadratic Discriminant Analysis

Christie Williams

September 30, 2009

Contents

1	Introduction	1
2	Classification of One-Dimensional Data	2
2.1	Linear Discriminant Analysis	2
2.1.1	Building the LDA Classifier	2
2.1.2	Results of One-Dimensional LDA Classification	4
2.2	Quadratic Discriminant Analysis	4
2.2.1	Building the QDA Classifier	5
2.2.2	Results of One-Dimensional QDA Classification	5
3	Classification of Two-Dimensional Data	6
3.1	Linear Discriminant Analysis	6
3.1.1	Building a Two-Dimensional LDA Classifier	7
3.1.2	Results of Two-Dimensional LDA Classification	7
3.1.3	Effects of Data Distribution on LDA Classification	8
3.2	Quadratic Discriminant Analysis	8
3.2.1	Building the QDA Classifier	8
3.2.2	Results of Two-Dimensional QDA Classification	9
3.2.3	Effects of Data Distribution on QDA Classification	9
4	Classification Over Real Data	10
4.1	Data Handling and Building the Classifiers	10
4.1.1	Preparing the Data	10
4.1.2	Building the Classifiers	11
4.1.3	Problems Encountered	14
4.2	Results and Observations	14
5	Conclusions	14

1 Introduction

This paper explores a couple different methods of classifying sets of data into a series of discrete classes. The two methods examined are linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA). In order to examine the pluses and minuses of each technique, each method is applied to both one-dimension and two-dimensional datasets, and the results are observed. In the two-dimensional datasets, each method is also applied to two types of datasets; one in which all classes are easily separable, and one in which datasets overlap. The results given by both LDA and QDA methods are then compared to attempt to determine what

LDA and QDA are each well-suited and badly-suited to doing. In addition, both methods are applied to a real-life dataset (a set of data dealing with classifying different samples of glass) in order to draw conclusions about the nature of the distribution of classes in the dataset.

2 Classification of One-Dimensional Data

The first aspect of this paper is the classification of one-dimensional data. This data was generated artificially (rather than from real-life observation) by creating three classes of data points, each from a Gaussian (Normal) distribution. For this sample data, the standard deviation for the three classes was kept constant at 0.1 (to keep data samples relatively close to the mean) and the means for the three classes generated were 1, 2, and 3, respectively. The code used to generate the data is as follows:

```
buildTrain1d <- function(sets=1, num=10, mu=0, sd=1)
{
  samples <- c()
  for (iter in 1:(sets))
  {
    samples <- cbind(samples, rnorm(num, mean=mu[iter], sd=sd))
  }
  return(samples)
}
```

The `sets` parameter specifies how many different classes should be generated (in the case of this experiment, three), the `num` parameter specifies how many samples should be taken from each class (in this case ten), and `mu` and `sd` are the mean and standard deviation for each set, respectively. (This method assumes a single standard deviation for all sets, but allows a series of different means. To make it more generally applicable, it would ideally allow a series of standard deviation values as well.) The datasets generated by a call to this function using the above means and standard deviation can be seen in the first graph in figures 1 and 2.

2.1 Linear Discriminant Analysis

Once the training data was generated, the experiment continued by generating the discriminant functions needed for an LDA analysis of test data. Each stage of this process is graphically illustrated in figure 1. All graphs were plotted using the `matplot` R command; a sample is as follows:

```
matplot(test1d, px_c, type="l", xlab="Test Data Value", ylab="P(x|C=k)", main=
  "Probability Sample is From Each Class")
legend("topleft", c("Class 1", "Class 2", "Class 3"), lty=c(1,2,3), col=c("
  black", "red", "green"))
```

2.1.1 Building the LDA Classifier

Test data for this experiment was created by simply creating a sequence of values between 0 and 4 (as the training classes centered around 1, 2, and 3, this allowed for some “extra” data that was slightly outside of the training range to be included in the testing sample). This sequence of data was generated with the R code:

```
test1d <- as.matrix(seq(0, 4, len=100))
```

The first step to building an LDA classifier is to calculate the three curves (one for each class) for $P(x - C=k)$ over the test data. This value is part of the final discriminant calculation, and represents the probability of x (an arbitrary test data value) given that the class is k (in this case, either 1, 2, or 3). The three curves can be seen in the second graph in figure 1. These values are calculated by solving a Gaussian equation for the probability of the test data given the mean and standard deviation for the desired class from the training data. The R code used to do this (for an arbitrary number of dimensions) is:

```

gaussian1d <- function(x, mu=matrix(0,nrow(x),1), sigma=diag(1, nrow(x)))
{
  x <- as.matrix(x)
  mu <- as.matrix(mu)
  sigma <- as.matrix(sigma)

  normConstant <- 1/(2 * pi) ^ (nrow(x)/2) * sqrt(det(sigma))
  invSigma <- solve(sigma)

  xZeroMean <- x - matrix(rep(mu, ncol(x)), nrow(x), ncol(x))

  return(normConstant * exp(-1/2 * colSums(t(t(xZeroMean) %*% invSigma)
    * xZeroMean)))
}

```

`x` is the set of data values for which probability is to be determined, and `mu` and `sigma` are matrices of mean and standard deviation values for the training data for one of the given classes. This code is a slightly modified version of code presented by Dr. Anderson in class [2].

Another part of the LDA calculation is $P(x)$, the probability of there being a data point for any class at a given value of x . This can be found using the R code:

```
px <- ((px_c[,1] * prob1) + (px_c[,2] * prob2) + (px_c[,3] * prob3))
```

`px_c` is the probability of x given that class is C from the previous calculation, and `prob1`, `prob2`, and `prob3` are the probability that data is from each of the three classes. In this case, since each class had 10 samples, all three probabilities are $1/3$. The output of this calculation can be seen in the third graph in figure 1.

The last piece of data needed to build the LDA classifier is $P(C=k | x)$, or the probability that the class equals k (either 1, 2, or 3) given a value x . This equation is arrived at by applying Bayes' Rule to the curves calculated in the second graph ($P(x | C=k)$). This curve can be calculated using the equation:

$$p(C = k|x) = \frac{p(x|C = k)p(C = k)}{p(x)} \quad (1)$$

In R code, this calculation is done by

```
pc_x <- (px_c[,1] * prob1) / px)
```

where `pc_k` is $P(C=k | x)$, `px_c` is $P(x | C=k)$ from the second step above, `prob` is the probability that the class is k , and `px` is the probability of a data sample from any class (given by the third step above). This is graphed for all three classes, producing three curves as seen in the fourth graph in figure 1.

From the above data, the discriminant functions for each class can then be calculated as follows:

```

makeLDADiscFunc <- function(mean, sigma, prior)
{
  sigmaInv <- solve(sigma)
  mean <- as.matrix(mean)
  function(X)
  {
    X <- as.matrix(X)

    mean <- matrix(mean, nrow(X), ncol(X), byrow=TRUE)
    diff <- X - matrix(mean, nrow(X), ncol(X), byrow=TRUE)

    rowSums((X %*% sigmaInv * mean) - (0.5 * mean %*% sigmaInv *
      mean) + log(prior))
  }
}

```

This method takes as parameters the mean and sigma for the given class, and the prior (the probability that any given data point is from the given class). In this case, the prior is always 1/3 because there are an equal number of samples in the training data for each of the three classes. Sigma is represented by a covariance matrix; for the LDA, it is a weighted average of the three covariance matrices (one for each class) calculated by:

```
avgCov <- cov1 * (1/3) + cov2 * (1/3) + cov3 * (1/3)
```

Once again, since the odds of a sample being from any class is equal, the weights are 1/3 for each covariance matrix. Because there is only one covariance matrix, the full discriminant function is simplified to the one used above, as several terms common to both sides can be dropped. The three discriminant functions can be seen in the fifth graph in figure 1.

Once the discriminant functions have been determined, it is a simple matter to classify test data points by determining which of the three determinant functions has the highest value for any given point. This is done with the simple R code:

```
solution <- (apply(cbind(lda1(test1d), lda2(test1d), lda3(test1d)), 1, which.max))
```

This code snippet applies the R `which.max` function to the three discriminants to determine which outputs the largest value for each testing data point. Results can be seen in graph six in figure 1.

2.1.2 Results of One-Dimensional LDA Classification

For the dataset used in this experiment, the LDA classifier did quite well. The boundaries between the three classes occurred quite close to the “actual” midpoints (roughly 1.5 and 2.5), and there were no outlying data points (for example on the low or high end) that were classified incorrectly. Part of the reason for this is likely that the datasets were easily separable; each class had a small standard deviation (0.1) compared to the distance between the means (each class’s mean was separated from the other two by 1), so that there was no overlap in training data. Because of this, linear discriminants are adequate to divide the data correctly.

One interesting aspect of the results of this experiment is the three discriminant functions produced. While intuitively it would seem that the “1” class’s discriminant should have a negative slope, it is in fact positive, though shallower than the slopes of the other two discriminants. This might possibly be due to a weighting of training data more towards the right hand side of the range of data, but this seems unlikely as this same pattern manifested throughout various trials of the classifier with randomly-generated data.

Another oddity in the results produced by this experiment is that the three probability distributions for data being from each sample, along with the overall probability of data belonging to a given class, are not evenly spaced (see graphs 2 and 3 in figure 1). Since the test data was an evenly-spaced sequence, this data should be evenly distributed across the entire range from 0 to 4. However, the class 2 and 3 curves overlap somewhat more than the class 1 and 2 curves do. This is likely caused by class 2 having a statistical (as opposed to the input it was generated with) mean slightly skewed high (i.e. above 2.0), as well as class 1 and class 3 most likely also not being directly centered over their “ideal” means either. These graphs show the way in which the classifier is affected by any “imperfections” in the training data.

One last point of interest is in the low percentages that all data has in graphs 2 and 3 (figure 1). Whereas the percentages would ideally be somewhere around (or possibly over) 1.0 at the height of the curves, the highest achieved is roughly 1/10 of that. While this does not affect the classifier’s accuracy, as all data is affected the same way, it does potentially indicate a problem. At this time, I believe the error may be in the Gaussian function (see code above, under the calculation of the second graph). This is an issue that should be pursued further.

2.2 Quadratic Discriminant Analysis

After running the LDA classifier over the data, the same data were used to generate and test a QDA classifier. The process of building and testing this classifier is graphically illustrated in figure 2. Once again, all graphs were plotted using the `matplot` R command, and test data was generated using the sequence of values between 0 and 4 as described above.

2.2.1 Building the QDA Classifier

The first three steps in building the QDA classifier are identical to those for building the LDA classifier; both the LDA and QDA classifiers use the same three calculations for $P(x | C=k)$, $P(x)$, and $P(C=k | x)$. The code to generate these is the same as for the LDA classifier above (see section 2.1, and the results can be seen in graphs 2 through 4 in figure 2. Note that these graphs were generated on a slightly different dataset than those referenced in the LDA section; as such, they are not identical.

The part of the QDA classifier that differs from the LDA classifier is the calculation of the discriminant functions. The LDA discriminant functions are actually a simplification of the QDA ones, arrived at by using a single average covariance matrix and dropping identical terms on all sides of the comparison. As such, the calculation is simplified to being linear in terms of x , rather than quadratic. The R code for the more complex quadratic discriminants is as follows:

```
makeQDADiscFunc <- function(mean, sigma, prior)
{
  sigmaInv <- solve(sigma)
  function(X)
  {
    X <- as.matrix(X)

    diff <- X - matrix(mean, nrow(X), ncol(X), byrow=TRUE)
    - 0.5 * log(det(sigma)) - 0.5 * rowSums(diff %*% sigmaInv *
      diff) + log(prior)
  }
}
```

As with the linear discriminant function, this method takes as parameters the mean and sigma for the given class, and the prior (the probability that any given data point is from the given class). The prior is once again always 1/3 because there are an equal number of samples in the training data for each of the three classes. Sigma is represented by a covariance matrix, which in this case is unique for each of the classes calculated. This code is a slightly modified version of code provided by Dr. Anderson in lecture slides [3]. The three quadratic discriminant functions can be seen in graph 5 in figure 2.

The classification of test data is done in the same manner for the QDA classifier as it is done for the LDA classifier, with the curve yielding the highest value for the data sample determining which class is the correct one. The R code to do this is essentially identical to that for LDA:

```
solution <- (apply(cbind(qda1(test1d), qda2(test1d), qda3(test1d)), 1, which.
  max))
```

The `qdaX` functions are the functions returned by the `makeQDADiscFunc` referenced above, and `test1d` is the test dataset. The output is one of the three classes; the results can be seen in graph 6 in figure 2.

2.2.2 Results of One-Dimensional QDA Classification

The classification results generated by the QDA classifier are virtually identical to those generated by the LDA classifier for these sets of training and test data. Because this data is so easily separable, there is not much difference between the simpler LDA and more complex (and more flexible) QDA classifier. Because no one class is undersampled in the training data, there does not appear to be significant overfitting being done by the QDA classifier. If there was, the LDA classifier would potentially provide a better fit, but in this case the two classifiers are roughly equivalent in accuracy.

Graphs 2 and 3 once again show the low percentage issue mentioned in the LDA section above; this is because these equations were calculated in the same way in both classifiers, and so exhibit the same (possibly erroneous) behavior. Similarly, the curves are once again moderately affected by variations in the training data, causing small shifts either left or right in the means. This can be seen by comparing either the two graph 2s or graph 3s in figure 1 and figure 2 as the same code was used to generate data samples for both.

3 Classification of Two-Dimensional Data

After experimenting on one-dimensional data, this report turns its focus to classifying two-dimensional data. In addition, this section of the report seeks to determine the ways that data distribution in the training data affects how well the classifier is able to predict later samples' class. In order to highlight these affects, the classifiers used in these experiments were tested on two sets of training data each; one set that was made up of well-defined classes, and one set that was made of data that was much more ambiguous in its class membership.

The datasets for this phase of experimentation were generated by using two Gaussian (Normal) distributions per class, to allow for greater variety in data values per class. In the “good” datasets (the ones that the classifiers are capable of doing well on), the two Gaussians for each class were close together; in addition, the sigma used was 0.2 to keep the generated points close to the given means. In the “bad” datasets (those the classifiers do poorly on), the two Gaussians were chosen to not be close to each other, and in addition the sigma used was doubled to 0.4 to give the data points a larger spread. The result is that the three classes' data overlap, making it much harder for the classifiers to determine which is the “correct” class. The R code to generate the “good” training dataset is:

```
#build the datasets (2 sample sets will be close together for 'good' fit)
means <- list(matrix(c(1,1,1,2), 2, 2, byrow=TRUE),
               matrix(c(3,1,2.5,1), 2, 2, byrow=TRUE),
               matrix(c(3,3,3,2), 2, 2, byrow=TRUE))

std <- 0.2
train2d <- NULL
for (class in 1:3)
{
  mus <- means[[class]]
  train2d <- rbind(train2d, cbind(rnorm(10,mus[1,1], std), rnorm(10, mus
    [1,2], std)))
  train2d <- rbind(train2d, cbind(rnorm(10,mus[2,1], std), rnorm(10, mus
    [2,2], std)))
}

classes <- c(rep(1, 20), rep(2, 20), rep(3, 20))
```

and the code used to generate the “bad” dataset is identical but with `means` and `std` defined as follows:

```
means <- list(matrix(c(1,1,2,2), 2, 2, byrow=TRUE),
               matrix(c(3,1,1,2), 2, 2, byrow=TRUE),
               matrix(c(3,3,2,1), 2, 2, byrow=TRUE))

std <- 0.4
```

This code is a slightly modified version of code provided by Dr. Anderson on the assignment page [1]. The datasets generated can be seen in the first graphs in figures 3, 4, 5, and 6; the first two are for the “good” dataset and the second two are for the “bad” dataset. Test data was generated with the code

```
testx <-seq(0, 4, len=100)
testy <- testx
test2d <- expand.grid(testx, testy)
```

which creates a series of all possible combinations of x and y values defined by the given sequence.

3.1 Linear Discriminant Analysis

The first classifier built for the two-dimensional data was an LDA classifier. The series of steps was similar to that used to build the one-dimensional LDA classifier, but some of the code had to be adapted to deal with multi-dimensional data. Details of the procedure are presented below, and graphs for each stage of the

process can be seen in figures 3, 4, 5, and 6. All graphs except for the initial data samples (graph 1) were plotted using the `persp3d` R command; a sample is as follows:

```
persp3d(z=px_c1, col="black", xlab="x", ylab="y", zlab="P(x|C=k)", main="
  Probability Sample is From Each Class")
persp3d(z=px_c2, col="red", add=TRUE)
persp3d(z=px_c3, col="green", add=TRUE)
```

In all graphs with multiple surfaces for different classes, the black surface represents class 1, the red surface represents class 2, and the green surface represents class 3.

3.1.1 Building a Two-Dimensional LDA Classifier

The first step in building the LDA classifier, calculating the curves for $P(x | C=k)$, is the same as that for the one-dimensional version (see section 2.1); this is due to the Gaussian function being created such that it can handle samples with arbitrary dimensionality. The only difference is that means and standard deviations are no longer single values; instead, they are matrices (in this case 1×2 and 2×2 , respectively) determined by the dimensionality of the sample data. The code for setting up these matrices is:

```
sigma1 <- matrix(c(sd(train2d[1:20,1]), 0, 0, sd(train2d[1:20,2])), 2, 2)
sigma2 <- matrix(c(sd(train2d[21:40,1]), 0, 0, sd(train2d[21:40,2])), 2, 2)
sigma3 <- matrix(c(sd(train2d[41:60,1]), 0, 0, sd(train2d[41:60,2])), 2, 2)
mu1 <- c(mean(train2d[1:20,1]), mean(train2d[1:20,2]))
mu2 <- c(mean(train2d[21:40,1]), mean(train2d[21:40,2]))
mu3 <- c(mean(train2d[41:60,1]), mean(train2d[41:60,2]))
```

where `train2d` is a 60×2 matrix containing 60 data points with an x and y coordinate for each. The curves produced by running each of these samples through the Gaussian function can be seen in graph 2 in figures 3 and 5.

Secondly, the $P(x)$ must be calculated for the testing data; the code to do this is identical to that used in the one-dimensional LDA classifier (see section 2.1), but is calculated automatically over both dimensions by R. The two-dimensional version can be seen in graph 3 in figures 3 and 5.

Next, $P(C=k | x)$ must be calculated. Once again, the two-dimensional version is the same as the one-dimensional version in section 2.1; the only change is that it is calculated over both dimensions. Graph 4 in figures 3 and 5 show the two-dimensional version (the relevant part of the graph is which surface is “highest” at a given point; the overlapping surfaces “underneath” are not significant and are an artifact of the graphics rendering).

Given these calculations, it is now possible to determine the discriminate functions for each class. As in previous parts of the calculation, the two-dimensional version is identical in R code to the one-dimensional version described in section 2.1. Graphs of the two-dimensional version’s output can be seen in graph 5 in figures 3 and 5.

At this point, the classification can be done by taking the maximum of the three discriminant functions at any given point. The two-dimensional application of this is the same as the one-dimensional version outlined in section 2.1, and the resulting graph can be seen in the last (sixth) graph in figures 3 and 5.

3.1.2 Results of Two-Dimensional LDA Classification

This section deals only with the results of running the LDA classification on the “good” dataset; discussion of the “bad” dataset is contained in the next section.

For the “good” dataset, the LDA Classification works quite well. In much the same way that the easily separable classes in the one-dimensional data lent themselves to being easily classified, the “good” dataset contains three distinct classes that can be easily divided by linear discriminants (planes, in the two-dimensional realm). As can be seen in graphs 2 through 4 in figure 3, there are three distinct “points” of data, one for each class. Even looking at the discriminant functions (graph 5 in figure 3) it is easy to see how the discriminants match up with approximate boundaries between classes in the training data.

However, one interesting aspect of this particular dataset is the large “blank” area that is classified as belonging to class 1. While it is possible that data that falls in that region is in fact a part of class 1, it could

conceivably be a part of class 3 as well. It is also possible that (assuming that the training data represented real observations rather than manufactured data) this gap is significant, and blindly classifying it as one of the existing classes is not the correct course of action. This is one of the downsides of simple classification; there is no provision for a certainty value being attached to the classification, meaning that a data point in the middle of this “gap” is assumed to be as sure as one centered near the mean of class 1.

Another aspect of the two-dimensional LDA classification is the recurring problem of the scale on graphs 2 and 3, which may be related to a problem with the Gaussian function as described in the one-dimensional results.

3.1.3 Effects of Data Distribution on LDA Classification

While the LDA classifier does well on well-behaved, easily separable classes, real data is often not this well-defined. In order to test how well the LDA classifier does on more realistic data, it was also run on a set of data that was much noisier (see graph 1 in figure 5, referred to above as the “bad” dataset). This data was generated with a higher value of sigma (0.4 instead of 0.2) and each class was comprised of two distinct Gaussians, with means centered far apart from each other in order to increase the variation in values represented in each class.

Clearly, the classifier does not do very well on this dataset. While the “corners” of the final prediction model (see graph 6 in figure 5) are not bad predictions, the center of the model is a different story. While one class is somewhat more likely, none of the classes really stand out as the “correct” solution and in particular some of the class 1 data near the center and some of the class 2 data in the upper left corner are severely mis-categorized. The main feature that makes this dataset significantly harder to classify is that the three classes are more or less merged together, rather than being distinct as in the “good” dataset. The second and third graphs in figure 5 show the problem; rather than three distinctive spikes of probability, one for each class, there is one large spike in the center that is a merger of all three classes. The fourth graph also shows the ambiguous nature of the classes; whereas the “good” dataset showed clear delineations between the three classes, this dataset produces overlapping curves that do not bear much resemblance to the original data.

Interestingly, the final classification model looks relatively similar to that produced for the “good” data, although the section for class 3 is somewhat larger and that of class 2 has shrunk. This appears to be due mostly to where the three classes are each most concentrated; class 2 appears to have the smallest “predicted” space because it contains two relatively isolated groups of data points whereas the other two classes have data more evenly spaced out. This may lead to a stronger weighting nearer to one of the two means, leading to a stronger presence in the final classifier for classes 1 and 3.

3.2 Quadratic Discriminant Analysis

After generating and testing the LDA classifier on these two data sets, a QDA classifier was built and tested using the same two sets of data (seen in graph 1 in figures 4 and 6). The process was similar to that for the LDA classifier; the graphs documenting the stages can be seen in figures 4 and 6. The first graph in each figure was plotted using the `matplot` R command; the rest were plotted using the `persp3d` R command as described in the previous section on LDA.

3.2.1 Building the QDA Classifier

The steps to building the QDA classifier were very similar to both the one-dimensional version of the QDA classifier (see section 2.2.1) as well as the two-dimensional version of the LDA classifier (see section 3.1). This is because both the LDA and QDA classifiers use the same three calculations for $P(x | C=k)$, $P(x)$, and $P(C=k | x)$, and most of the R code can be used interchangeably for one- or multi-dimensional calculations. Graphs 2 through 4 in figures 4 and 6 show the results of these three calculations over the two-dimensional datasets.

The QDA classifier diverges from the LDA classifier in how the discriminant functions are calculated. As in the one-dimensional version, the difference is that the QDA discriminant functions use unique covariance matrices, rather than an average one drawn from all three classes. The code to generate the covariance

matrices is the same as that for the LDA classifier, but the three covariance matrices are simply used directly, rather than averaged first. The code is as follows:

```
mean1 <- colMeans(as.matrix(train2d[1:20,]))
cov1 <- cov(as.matrix(train2d[1:20,]))
mean2 <- colMeans(as.matrix(train2d[21:40,]))
cov2 <- cov(as.matrix(train2d[21:40,]))
mean3 <- colMeans(as.matrix(train2d[41:60,]))
cov3 <- cov(as.matrix(train2d[41:60,]))
```

```
qda1 <- makeQDADiscFunc(mean1, cov1, (1/3))
qda2 <- makeQDADiscFunc(mean2, cov2, (1/3))
qda3 <- makeQDADiscFunc(mean3, cov3, (1/3))
```

The `makeQDADiscFunc` function is the same one described above in section 2.2.1. The three quadratic discriminant functions can be seen in graph 5 in figures 4 and 6.

The final classification is done in the same way as in previous classifiers; the discriminant function yielding the highest value for a given point belong to the predicted class. The R code for this is identical to previous classifiers:

```
solution <- (apply(cbind(qda1(test2d), qda2(test2d), qda3(test2d)), 1, which.
max))
```

where `qda1`, `qda2`, and `qda3` are the QDA discriminant functions generated in the previous step. The final classification model can be seen in graph 6 in figures 4 and 6.

3.2.2 Results of Two-Dimensional QDA Classification

This section deals only with the results of running the QDA classification on the “good” dataset; discussion of the “bad” dataset is contained in the next section.

For the most part, the results from the QDA classifier are quite similar to the results for the LDA classifier over the “good” dataset. This is because the classes are so distinct that it is easy for the classifier to determine the boundaries with either the simpler (linear) or more complex (quadratic) models. Interestingly, overfitting does not appear to be a problem with this dataset, although it seemed as though it could be given the large gap around class 1’s data. Instead, the QDA classifier generated similar discriminants, which do not appear to have overfit the data. One consequence of this, however, is that, similar to the LDA classifier’s predictions, there is a large range of points for which the classifier will predict a class when there is not any training data in the area to solidly justify the decision. As with the LDA classifier, there is no confidence value associated with the final prediction, which means the classifier appears to be as “sure” of a prediction in the area where there are gaps in data as in areas that are near to the means of the classes of data present. This is the sort of data distribution that shows why confidence values can be a useful addition to classification schemes.

In addition, this set of results, like the others, suffers from the potential Gaussian function error leading to low percentage values for graphs 2 and 3 (see figure 4).

3.2.3 Effects of Data Distribution on QDA Classification

As for the LDA version of the classifier, testing the classifier on “artificial”, well-behaved data is not enough. It is also important to see how well it performs on messier, “real-world” data (see graph 1 in figure 6). This data is referred to as the “bad” dataset above, and was generated with a higher value of sigma (0.4 instead of 0.2) and each class was comprised of two distinct Gaussians (this is the same “bad” dataset used to test the LDA classifier).

As with the LDA classifier, the QDA classifier is not able to easily predict classes when given a set of noisy data. In areas where one class dominates (for example, in the lower right corner where data are mainly from class 2, or in the upper right corner where data are mainly from class 3) the predictions are relatively accurate, but in the central area, where all three classes overlap, the classifier does much worse. This is because, even with the added complexity of quadratic discriminant functions, the data is too varied for a simple surface to correctly separate the classes apart.

One further experiment that it would be interesting to conduct would be to have only one class that is “noisy”, and have the other two classes be similar to those generated in the “good” dataset. Similarly, it would also be interesting to have the “bad” class be composed of more discrete sets of data, rather than one large, wide-ranging set. These further experiments could be used to explore what it is about the data that causes the classifiers such trouble; for example, is it that the classes are too “general”, or that they overlap each other, or simply that there are no good locations to separate the data between classes until a very large number of dimensions are used to build the discriminant functions? Further experimentation with various “bad” datasets would be enlightening in this area of exploration.

4 Classification Over Real Data

After building and testing various LDA and QDA classifiers over “manufactured” sets of one- and two-dimensional data, the next part of this project was examining using the same LDA and QDA techniques to build classifiers for a real set of data. The dataset chosen for this set of experiments was a set of 214 samples of various kinds of glass, with a category for type of glass (such as building windows, car windows, containers, tableware, etc.) that could be predicted from the other attributes (primarily contents of various minerals, as well as the glass’s refractive index). This dataset came from the UCI Machine Learning Repository and is referred to there as the “glass” dataset [5]. This source of data was chosen as it was a relatively small, yet still reasonably-sized dataset (some other datasets contained thousands of entries, or a few dozen; small datasets did not seem as though they would yield enough samples for useful classification results, and large datasets would potentially be too time-consuming to run many times in testing). In addition, the classes in the dataset were fairly complex; there were seven different possible types of glass, and to make the data even more interesting, one class did not have any values present (class 4, specifically). It seemed as though this relatively complex classification environment would give the classifying algorithms a good workout, without the added complexity of needing to transform the input data in order to make it usable (i.e., there were no input fields that depended on each other, or arbitrary integers used as discrete values other than the final glass type parameter).

4.1 Data Handling and Building the Classifiers

4.1.1 Preparing the Data

The first step necessary for using a real set of data was to read in and prepare the data in R. Reading in the file was simple; this can be done in R with the code

```
glass <- read.table("glass.data", sep=",")
```

In this case, it is necessary to specify the separator, which was a comma for this dataset. One simple modification to the data was to trim off the first column; it was not a feature, but simply an index of each sample, and as such, irrelevant to the classifiers (although as the data was sorted, it could be argued that it may have been an effective predictor. However, it is an artificial piece of information rather than one drawn from the samples themselves). The R code to do this was

```
glass <- glass[,2:11]
```

Once the dataset had been properly prepared, column names were added. This is not strictly necessary to the classification but it makes debugging and visualization of the data (for example, with the `pairs()` R call) easier. The R code for this was

```
glassNames <- c("Refraction Index", "Sodium", "Magnesium", "Aluminum", "Silicon",  
               "Potassium", "Calcium", "Barium", "Iron", "Type")  
colnames(glass) <- glassNames
```

After this, the next step was to separate the data into training and test sets. For this experiment, 80% of the data was used for training, with the remaining 20% used for testing. Data was randomly chosen to be in the training and test categories; while some classes had more samples than others, they were not skewed enough to warrant specifically choosing a proportionate number of samples from each class. Code to divide the data is as follows:

```
temp_data <- sample(nrow(glass))
split <- round(nrow(glass) * 0.8)
trainX <- glass[temp_data[1:split], 1:9]
trainT <- glass[temp_data[1:split], 10, drop=FALSE]
testX <- glass[temp_data[(split+1):nrow(glass)], 1:9]
testT <- glass[temp_data[(split+1):nrow(glass)], 10, drop=FALSE]
```

The `trainX` and `testX` matrices are the attributes used for prediction, and the `trainT` and `testT` matrices are the corresponding classes by sample (ordered in the same randomly-chosen order as the prediction attributes). They are used to evaluate how well the classifiers did for both the training and testing data. At this point, the data is ready to be used to build the LDA and QDA classifiers.

4.1.2 Building the Classifiers

The steps in building the LDA and QDA classifiers for real data are essentially the same as those for the “artificial” data; full code for the QDA classifier (the LDA classifier is essentially identical with the exception of the discriminant functions) can be found at the end of this section, but the discussion will focus primarily on the changes that were needed in order to handle the irregularities of an actual dataset. One other detail to note about this code that is different from the previous classifiers is that training and test data were both run through the classifiers, meaning that each step is duplicated for both the training set and the testing set. Also, unlike the “artificial” data, this data needed to be standardized; this was done with the R code

```
standardize <- makeStandardize(trainX)
trainX <- standardize(trainX)
testX <- standardize(testX)
```

with the `makeStandardize` function defined as follows:

```
makeStandardize <- function(X) {
  if (missing(X)) {
    cat("Usage:
      standardize <- makeStandardize(X) ## X is nSamples x nDimensions
      Xs <- standardize(X)
      X2s <- standardize(X2)\n")
  }
  return(invisible())
}
## X is nSamples x nDimensions
mu <- colMeans(X)
sigma <- sd(X) ##sd should be named colSDs

function(newX) {
  nr <- nrow(newX)
  nc <- ncol(newX)
  (newX - matrix(mu, nr, nc, byrow=TRUE)) / matrix(sigma, nr, nc, byrow=TRUE)
}
}
```

This code is a slightly modified version of code provided by Dr. Anderson on lecture slides [4].

The first steps were to calculate the three pieces of data needed to calculate the discriminants: $P(x | C=k)$ for each class, $P(x)$ for the dataset as a whole, and $P(C=k | x)$ for each class. As the individual pieces were not being graphed, and there were more classes to deal with, many of the computations were done in loops, as can be seen in the code below. This allowed for the data dealing with the larger number of classes to be handled with less code, although ideally more advanced matrices could be used as they are more efficient in R.

The next step was to calculate the discriminant functions; once again, the code was similar to the one- and two-dimensional code, and the only difference between the QDA and LDA versions was that the LDA determinants were calculated with the same covariance matrix, averaged between all classes, and the QDA

determinants each used their own covariance matrix. The functions for making the discriminants are the same as those referenced and detailed in earlier sections.

Full code for building the QDA classifier can be seen below:

```

means <- c()
sigmas <- c()

#create training means and sigmas for all classes
for (iter in 1:7) #7 possible classes
{
  #calculate the three parts of the discriminant functions for
  the data

  trainX_filtered <- trainX[trainT==iter,] #select all samples
  for this class

  if (nrow(trainX_filtered) < 1)
  {next}

  mu <- mean(trainX_filtered)

  #hack to fix case where all of a value are 0
  raw_sigma <- sd(trainX_filtered) + 0.00001

  means <- cbind(means, mu)
  sigmas <- cbind(sigmas, raw_sigma)
}
px_c_train <- c()
px_c_test <- c()
prob_train <- c()
prob_test <- c()
pc_x_train <- c()
pc_x_test <- c()
disc_train <- c()
disc_test <- c()

for (iter in 1:6) #skip empty class 4
{
  iter_class <- iter
  #skip empty class 4
  if (iter == 4)
  {iter_class <- iter + 1}

  trainX_filtered <- trainX[trainT==iter_class,] #select all
  samples for this class

  testX_filtered <- testX[testT==iter_class,] #select all
  samples for this class

  prob_xc_train <- gaussian1d(t(trainX_filtered), as.matrix(
  means[,iter]), diag(sigmas[,iter], ncol(trainX_filtered),
  ncol(trainX_filtered)))
  prob_xc_test <- gaussian1d(t(testX_filtered), as.matrix(means
  [,iter]), diag(sigmas[,iter], ncol(testX_filtered), ncol(
  testX_filtered)))
}

```

```

px_c_train <- cbind(px_c_train, prob_xc_train)
px_c_test <- cbind(px_c_test, prob_xc_test)

prob_train <- cbind(prob_train, (nrow(trainX_filtered) / nrow(
  trainX)))
prob_test <- cbind(prob_test, (nrow(testX_filtered) / nrow(
  testX)))
}

px_train <- ((px_c_train[,1] * prob_train[,1]) + (px_c_train[,2] *
  prob_train[,2]) + (px_c_train[,3] * prob_train[,3]) + (px_c_train
  [,4] * prob_train[,4]) + (px_c_train[,5] * prob_train[,5]) + (px_c
  _train[,6] * prob_train[,6]))
px_test <- ((px_c_test[,1] * prob_test[,1]) + (px_c_test[,2] * prob_
  test[,2]) + (px_c_test[,3] * prob_test[,3]) + (px_c_test[,4] *
  prob_test[,4]) + (px_c_test[,5] * prob_test[,5]) + (px_c_test[,6]
  * prob_test[,6]))

for (iter in 1:6) #skip empty class 4
{
  iter_class <- iter
  #skip empty class 4
  if (iter == 4)
  {iter_class <- iter + 1}

  trainX_filtered <- trainX[trainT==iter_class,] #select all
    samples for this class
  testX_filtered <- testX[testT==iter_class,] #select all
    samples for this class

  pc_x_train <- cbind(pc_x_train, ((px_c_train[,iter] * prob_
    train[,iter]) / px_train))
  pc_x_test <- cbind(pc_x_test, ((px_c_test[,iter] * prob_test[,
    iter]) / px_test))

  qda_train <- makeQDADiscFunc(colMeans(as.matrix(trainX_
    filtered)), cov(as.matrix(trainX_filtered)+0.00001), prob_
    train[,iter])
  qda_test <- makeQDADiscFunc(colMeans(as.matrix(testX_filtered)
    ), cov(as.matrix(testX_filtered)+0.00001), prob_test[,iter
    ])
  disc_train <- cbind(disc_train, qda_train(trainX))
  disc_test <- cbind(disc_test, qda_test(testX))
}

#predicted class
solution_train <- (apply(cbind(disc_train[,1], disc_train[,2], disc_
  train[,3], disc_train[,5], disc_train[,6], disc_train[,7], 1,
  which.max)))
solution_test <- (apply(cbind(disc_test[,1], disc_test[,2], disc_test
  [,3], disc_test[,5], disc_test[,6], disc_test[,7], 1, which.max)))

```

4.1.3 Problems Encountered

Unlike the “manufactured” data, this data did not always conform nicely to the methods used in previous sections. Due to this, I was unfortunately not able to achieve working classifiers. In particular, three areas caused repeated difficulty: the class (class 4) which did not have any samples; the fact that a large number of values, particularly for Iron content, had values of 0 (meaning that it was quite possible to have a sampling in which every single value, and hence mean and standard deviation, were 0 as well); and the fact that not all classes had the same number of samples.

This last problem meant that some matrices, such as the covariance matrices, were not the same size for each sample. This meant that, for example, in the LDA classifier, there was no easy way (or I did not have time to discover one) to average the covariance matrices together. In addition, it made some of the methods used to aggregate values (such as the $P(x | C=k)$ values) fail, as R calls such as `cbind` assume all matrices are the same dimensionality and will pad or truncate ones that do not match.

The issue with some attributes returning 0 means and standard deviations was fixable with a small hack of adding 0.00001 to all values in the matrix; this prevented singularities when doing matrix manipulations without seriously modifying data values, but it is not an idea solution and there are likely better ones available.

The problem with class 4 having no samples primarily manifested itself in dealing with looping over all classes. Since it had no values, attempting to do any calculations on that class (such as determining probabilities) would result in errors as there was no data available. Due to this, various steps (commented in code in previous section) were necessary to make sure that that class was skipped where appropriate.

4.2 Results and Observations

Unfortunately, I was not able to achieve results from this experiment as I ran out of time to get the classifiers working. However, the results that I expected to see fall into one of two categories, which would say different things about the distribution of the data. I would expect to see either fairly distinct classes (shown by a relatively low error rate, and a relatively even error rate between the training and test data) or classes that overlap significantly (shown by poor classification rates, with a possibly better rate in the training than the test data). If the results had been more medium (for example, around an 85% to 90% correct rate), I would conclude that the data is likely somewhere between these two extremes; this would tend to imply some overlap, but with classes still fairly distinctive. However, if the rate was primarily bad in only a few classes, then I would draw the conclusion that while most classes were fairly distinct, those few perhaps had a larger overlap or were perhaps less well-defined (i.e., had a higher covariance).

In order to determine the error rate, the code I was planning to use was a ratio of correct predictions to number of predicted (either test or train) samples; this could be tracked relatively easily during the prediction phase by using the `trainT` and `testT` matrices containing the correct classes. In addition, I would graphically represent the classes for both the training and testing data, for each of the LDA and QDA classifiers. Each graph would show the actual data by class as well as the prediction. Sample code is as follows:

```
solution_test <- (apply(cbind(disc_test[,1], disc_test[,2], disc_test[,3],
  disc_test[,4], disc_test[,5], disc_test[,6], 1, which.max)))
matplot(seq(0,length(testT),by=1), c(testT, solution_test), type="p", pch=c
  (1,2), col=c("red", "green"), xlab="Test Data Index", ylab="Predicted
  Class", main="Predicted Class for Test Data")
```

5 Conclusions

Overall, this project served to illustrate the ways in which even relatively simple classifiers (i.e., LDA or QDA) can quite accurately predict the class of data, at least under conditions where the classes are relatively distinct. At the same time, however, it showed how important the distribution of data is; when the classes overlap to any degree, these classifiers quickly fail. Interestingly, there was not as much difference between the QDA and LDA classifiers as I would have expected; the QDA classifiers did not appear to overfit to

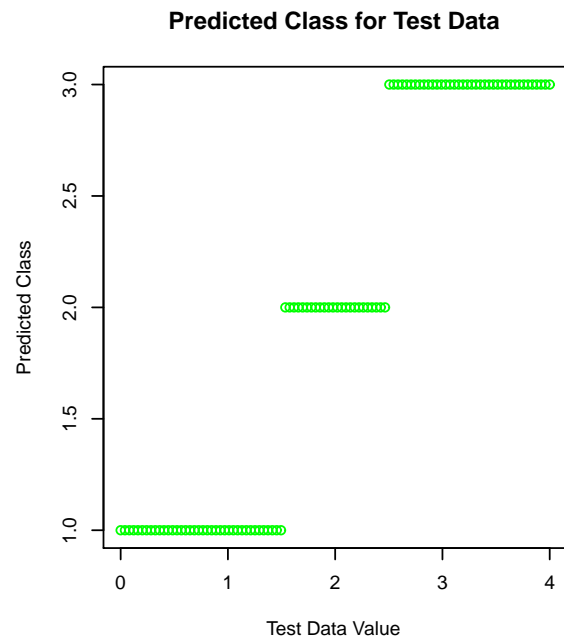
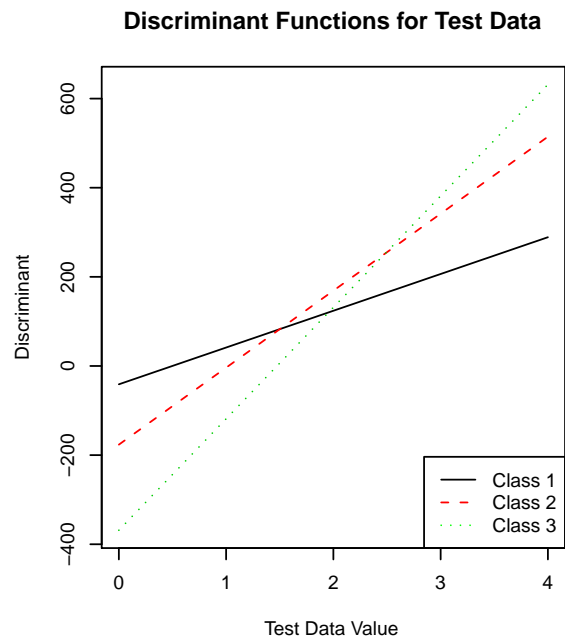
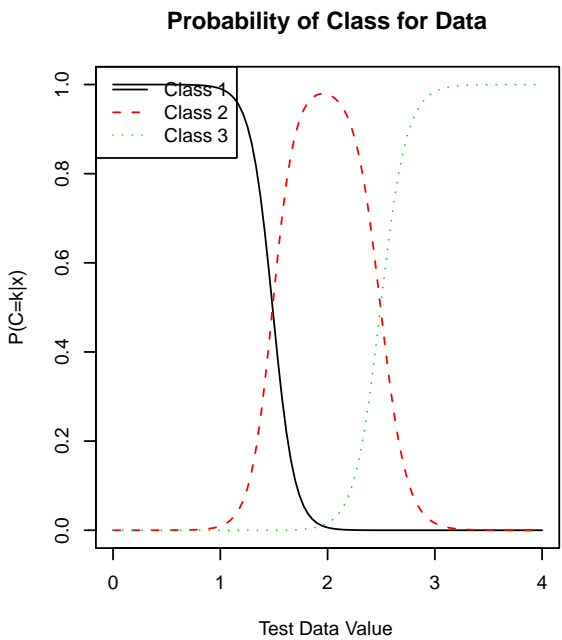
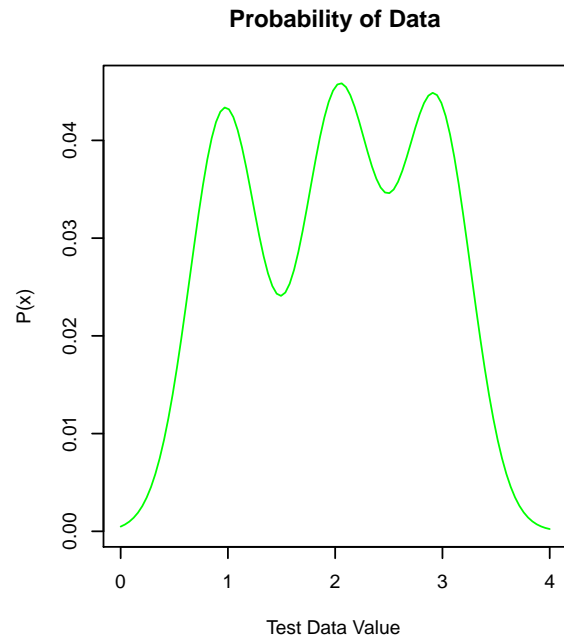
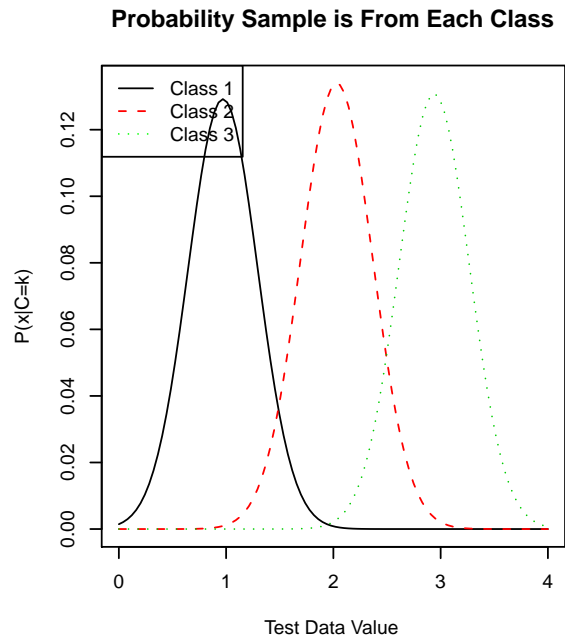
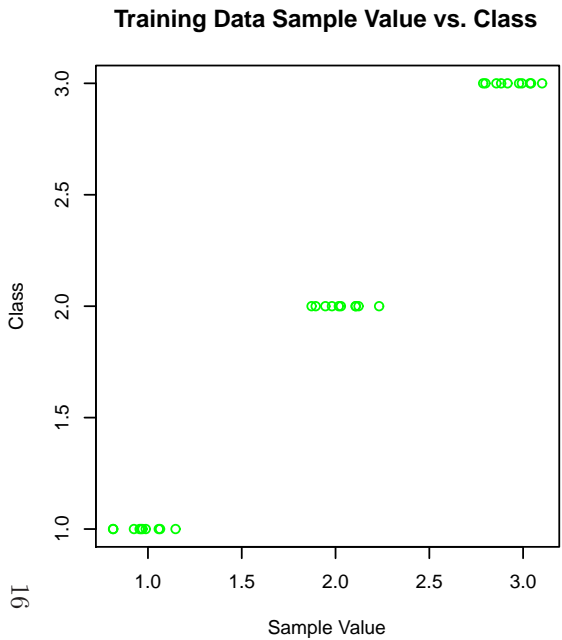
any large extent, and the LDA classifiers seemed to be close to as accurate despite being simpler. This is something that I would be interested in investigating further, and perhaps experimenting around with finding datasets that cause more overfitting or that require the added complexity of a QDA classifier.

The most difficult aspect of this project was dealing with the actual dataset. While the “artificial” data behaved nicely, and had such useful aspects as having the same number of samples from each class, real data is not that way. Trying to conform my methods for producing the classifiers to real-world glitches was difficult and frustrating as I encountered problems I had not even thought of earlier.

References

- [1] Anderson, C., *Assignment 3 Writeup*, <http://www.cs.colostate.edu/~anderson/cs545/assignments/assignment3.html>, 2009.
- [2] Anderson, C., *Lecture Notes - CS545: Bishop(2.3-2.5, 3.1)*, <http://www.cs.colostate.edu/~anderson/cs545/Lectures/week2day2/day2.pdf>, 2009.
- [3] Anderson, C., *Lecture Notes - CS545: Linear Models for Classification*, <http://www.cs.colostate.edu/~anderson/cs545/Lectures/week4day2/week4day2.pdf>, 2009.
- [4] Anderson, C., *Lecture Notes - CS545: Linear Modeling*, <http://www.cs.colostate.edu/~anderson/cs545/Lectures/week3day2/week3day2.pdf>, 2009.
- [5] UCI Machine Learning Repository, <http://archive.ics.uci.edu/ml/machine-learning-databases/glass/>, 1987.

Figure 1: Linear Discriminant Analysis (one dimension): Graph 1: one-dimensional training data; Graph 2: curves for probability of test data x given class (1, 2, or 3); Graph 3: probability of data in any class (test data); Graph 4: curves for probability of class (1, 2, or 3) given test data x ; Graph 5: LDA discriminant functions; Graph 6: predicted class for test data



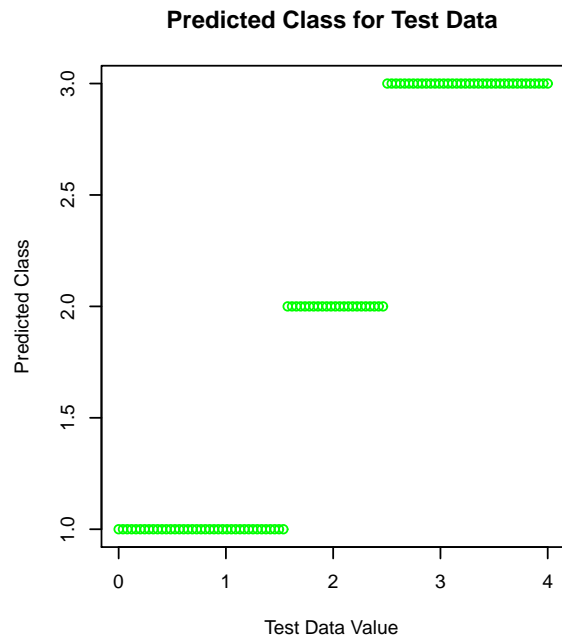
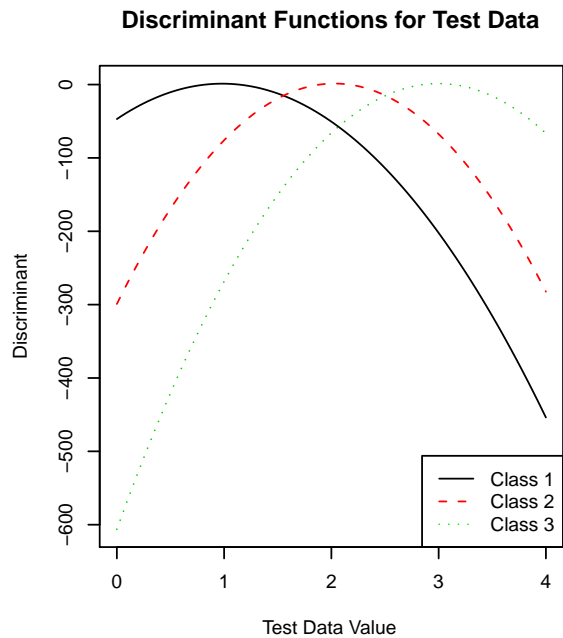
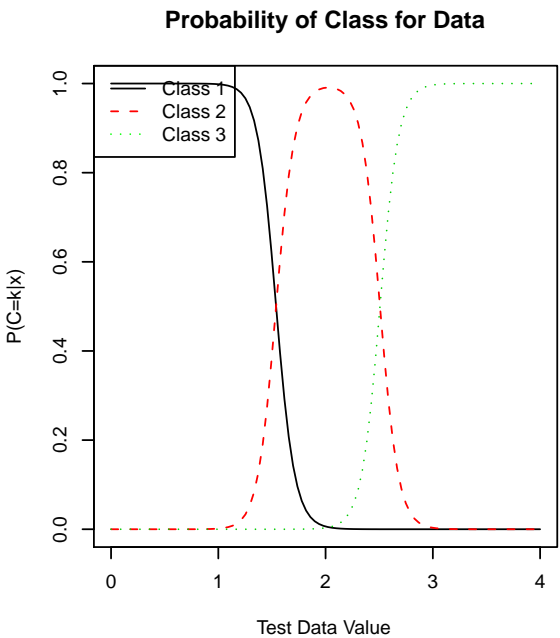
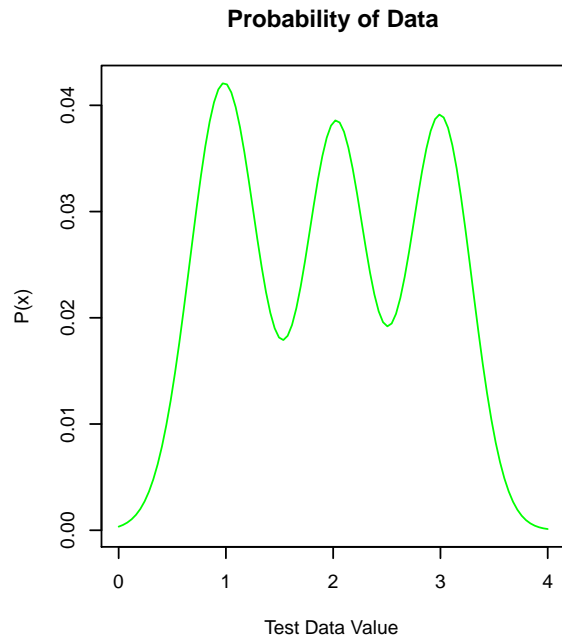
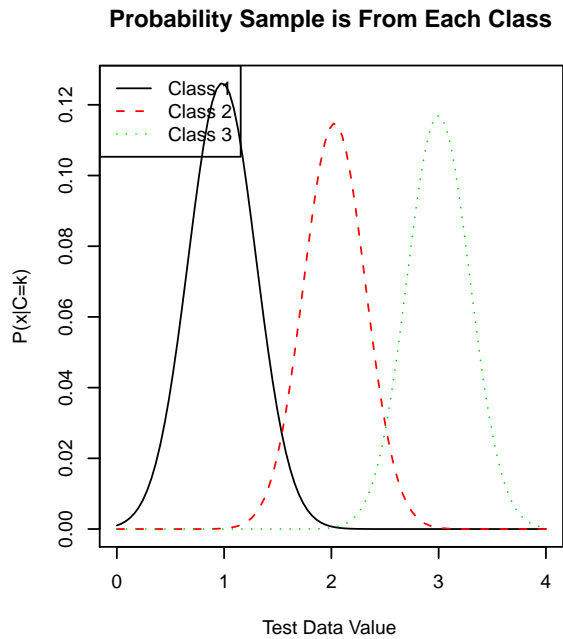
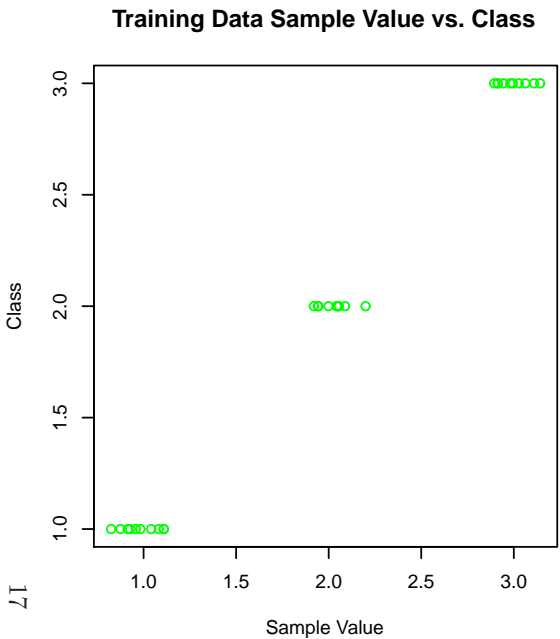


Figure 2: Quadratic Discriminant Analysis (one dimension): Graph 1: one-dimensional training data; Graph 2: curves for probability of test data x given class (1, 2, or 3); Graph 3: probability of data in any class (test data); Graph 4: curves for probability of class (1, 2, or 3) given test data x ; Graph 5: QDA discriminant functions; Graph 6: predicted class for test data

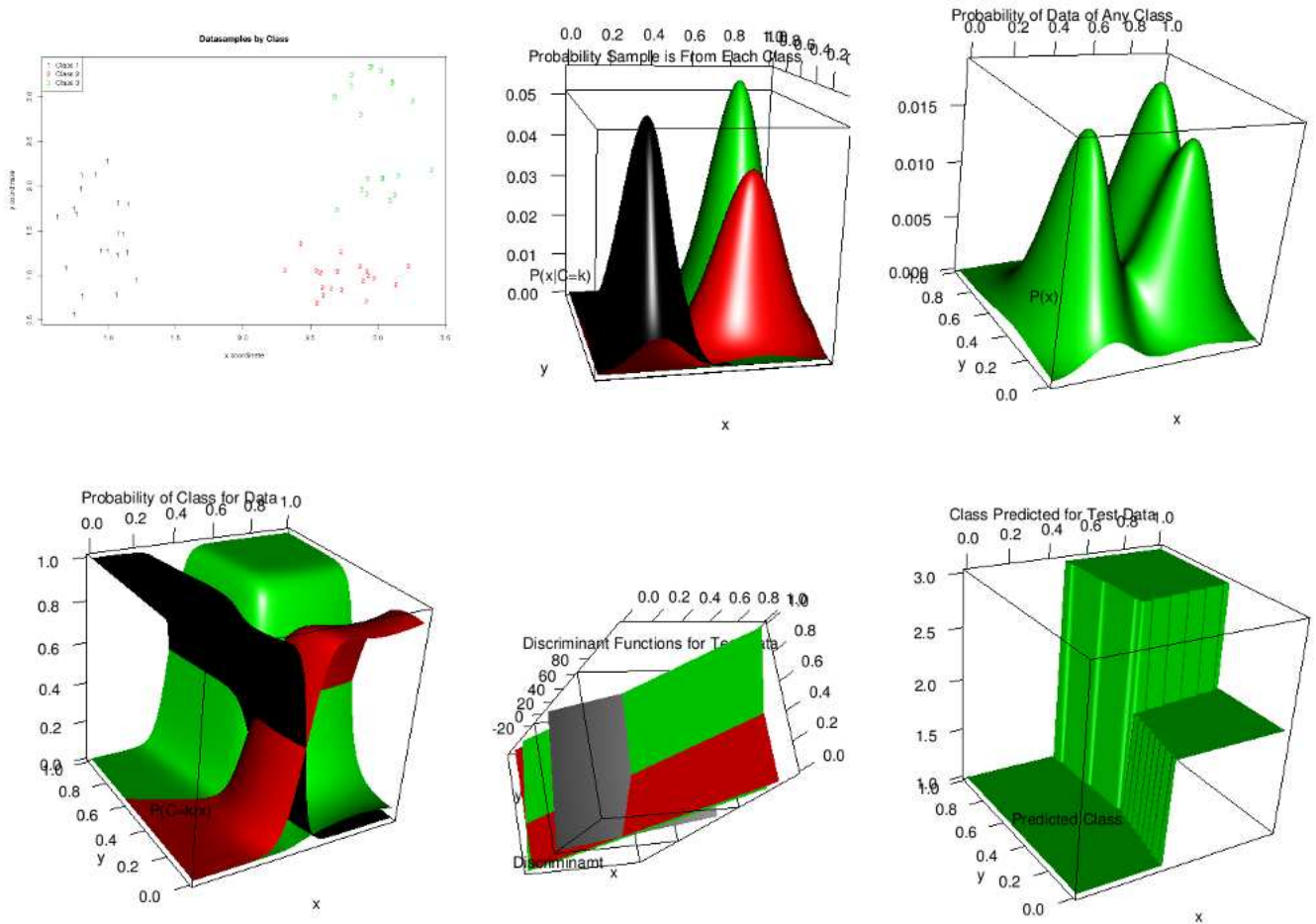


Figure 3: Linear Discriminant Analysis (two dimensions, “good” data): Graph 1: two-dimensional training data; Graph 2: curves for probability of test data x given class (1, 2, or 3); Graph 3: probability of data in any class (test data); Graph 4: curves for probability of class (1, 2, or 3) given test data x ; Graph 5: LDA discriminant functions; Graph 6: predicted class for test data

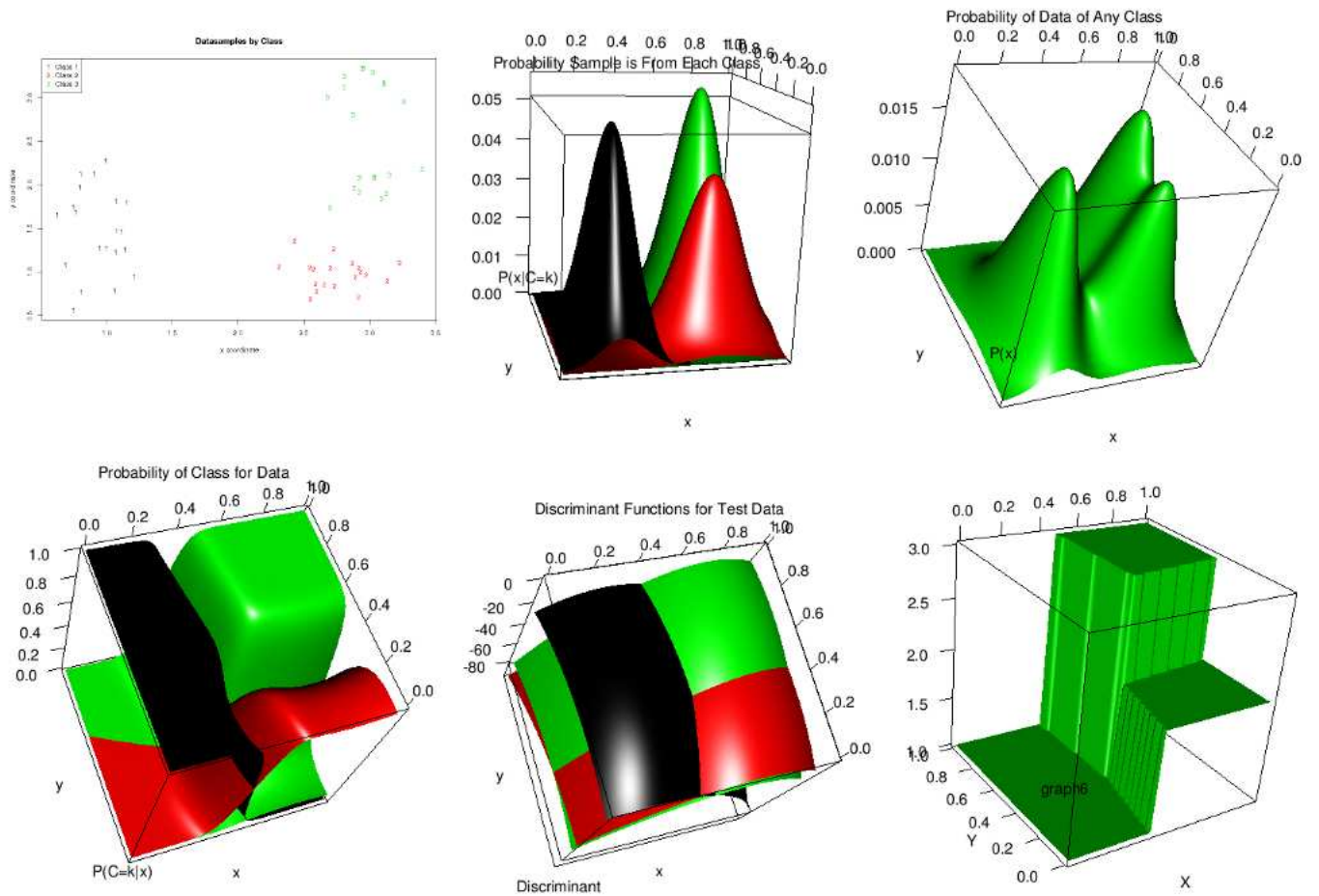


Figure 4: Quadratic Discriminant Analysis (two dimensions, “good” data): Graph 1: two-dimensional training data; Graph 2: curves for probability of test data x given class (1, 2, or 3); Graph 3: probability of data in any class (test data); Graph 4: curves for probability of class (1, 2, or 3) given test data x ; Graph 5: QDA discriminant functions; Graph 6: predicted class for test data

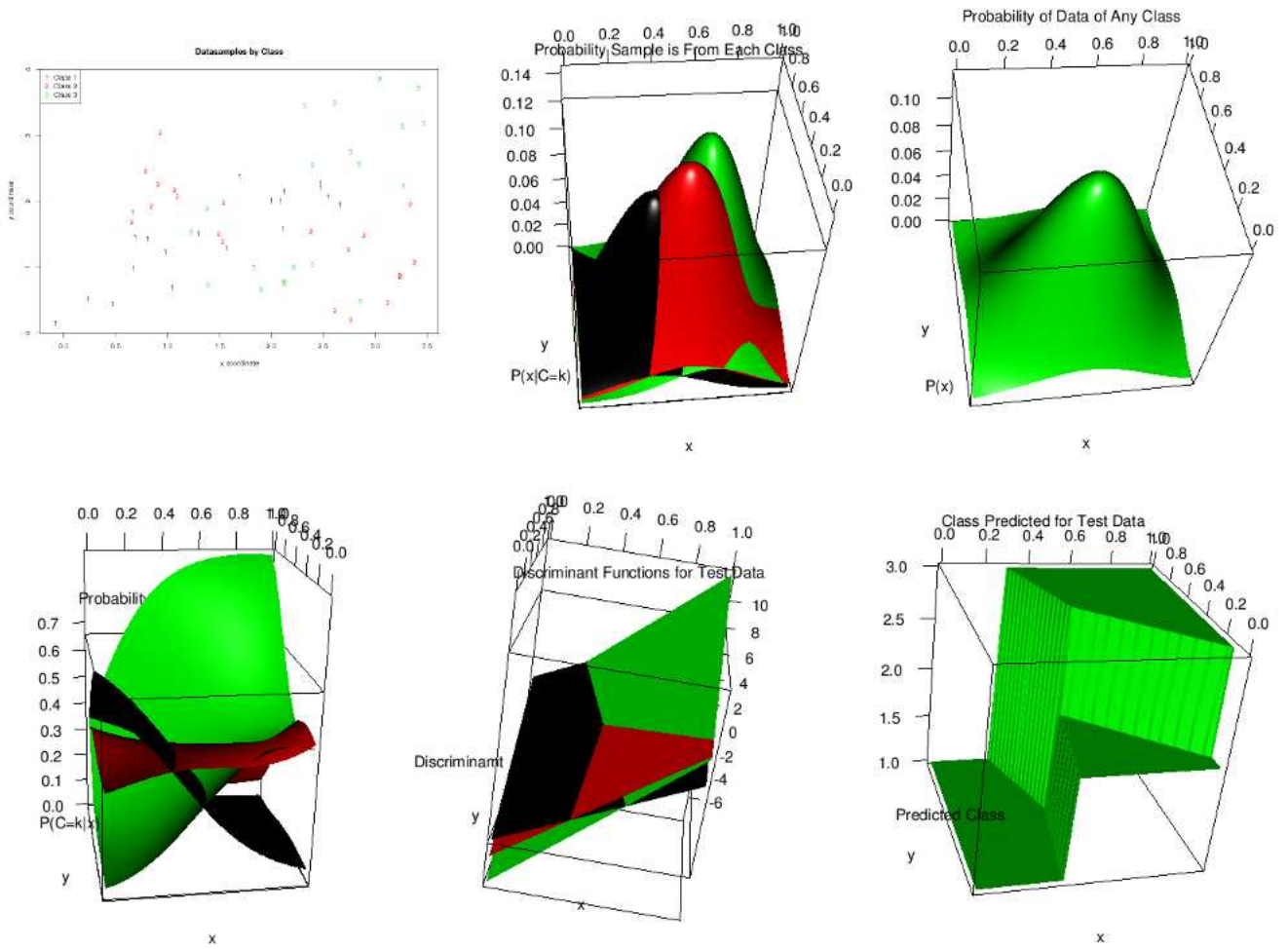


Figure 5: Linear Discriminant Analysis (two dimensions, "bad" data): Graph 1: two-dimensional training data; Graph 2: curves for probability of test data x given class (1, 2, or 3); Graph 3: probability of data in any class (test data); Graph 4: curves for probability of class (1, 2, or 3) given test data x ; Graph 5: LDA discriminant functions; Graph 6: predicted class for test data

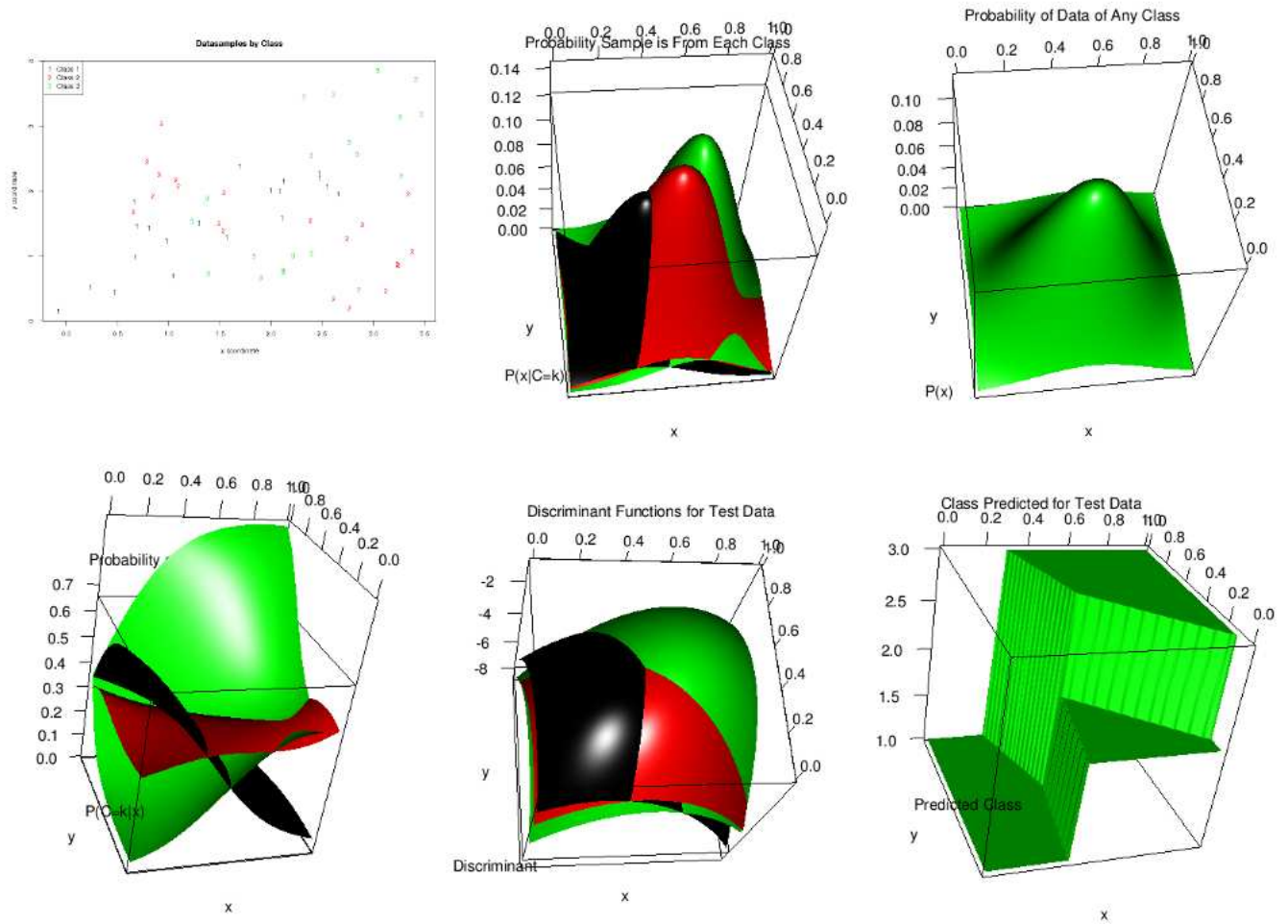


Figure 6: Quadratic Discriminant Analysis (two dimensions, "bad" data): Graph 1: two-dimensional training data; Graph 2: curves for probability of test data x given class (1, 2, or 3); Graph 3: probability of data in any class (test data); Graph 4: curves for probability of class (1, 2, or 3) given test data x ; Graph 5: QDA discriminant functions; Graph 6: predicted class for test data