SUPERVISED CLASSIFICATION USING DISCRIMINANT ANALYSIS

RECONNAISSANCE DES FORMES

Iacob Sergiu

11th March 2020

Contents

CHAPTER 1

INTRODUCTION

1.1 CONTEXT

This report is the result of a university assignment. It aims to prove that the student understood the motivation, goals and means of studying pattern recognition. Therefore, this is a way of summarizing a series of observations and experiments done on images containing shapes.

1.2 MOTIVATION

Classifying objects is a task that we must do every day. It is incredibly versatile and universally encountered: whether we should classify a thing as being healthy or not, a shape as being a circle or not, we must do it everyday. It is inevitable that classification tasks have made their way into Computer Science as well, in every domain. Having accurate ways of doing these kind of tasks is, therefore, widely needed, so we will focus on analysing some methods of doing this task.

1.3 GOALS

Our goal here is to analyse and classify instances from a dataset of points. Each point is labeled, therefore this is a supervised classification task. We will try to mathematically express relationship between the points' attributes (in our case, their positional values in the 2D space) that will help us conclude certain traits on the dataset. The conclusions will have to be as close as possible to the ground truth, so we will also be interested in "scoring" as well as possible with our algorithms.

CHAPTER 2

BINARISATION USING BAYES' THEOREM

2.1 VISUALISING DATA

The data we are going to use is present in a file of type `Rdata` and it consists of two parts: data for *training* and data for *testing*. Each dataset has a number of 75 instances (an instance being described by 2 *features*) which are divided (not equally) amongst 3 *classes*.

Having a *true class* assigned to each instance, we can treat this problem as a *supervised classification* problem. Moreover, we also have information about true *means* and *standard deviations*, so we can use these to make comparisons between our algorithms and the truth.

Let's take a look at how the data looks:

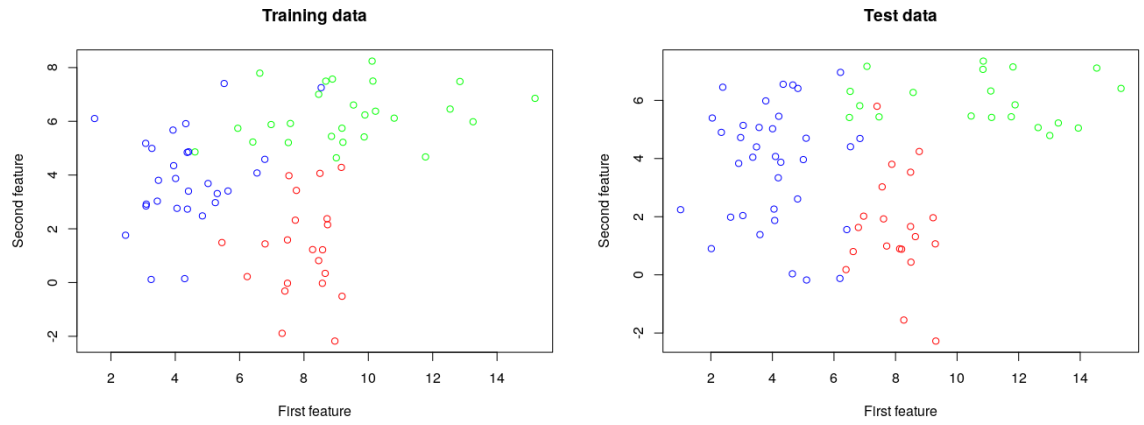

FIGURE 2.1
Plotted data. Train on left, test on right

```
data <- load(file='simul-2017.Rdata')
class_colors = c('red', 'green', 'blue')
couleur_app <- rep('red', n_app)
for (class_no in 1:3)
  couleur_app[classe_app == class_no] = class_colors[class_no]
plot (x_app, col=couleur_app, main='Training data', xlab='1st feature', ylab='2nd
    feature')
for (class_no in 1:3)
  couleur_app[classe_test == class_no] = class_colors[class_no]
plot (x_test, col=couleur_app, main='Test data', xlab='1st feature', ylab='2nd
    feature')
```

LISTING 2.1
Loading and visualising data

We can notice more blue dots, which would mean instances are not equally distributed amongst classes. Calculating the class a priori probabilities, that is indeed true:

```
# P(w) = probability of an instance to be in the class "w"
class_probs_app <- 1:3
class_probs_test <- 1:3
for (class_no in 1:3){
    class_probs_app[class_no] = sum(class_no == classe_app) / length(classe_app)
    class_probs_test[class_no] = sum(class_no == classe_test) / length(classe_test)
}
class_probs_app
class_probs_test
```

LISTING 2.2
A priori class probabilities

The script above gives us the values $P(red) = 0.28$, $P(green) = 0.3466667$ and $P(blue) = 0.3733333$ for the training dataset and $P(red) = 0.2666667$, $P(green) = 0.2666667$ and $P(blue) = 0.4666667$, for the test dataset. The values corresepond to red, green and blue classes, in this order.

2.2 ESTIMATING MEANS AND COVARIANCE

2.2.1 CALCULATING MEANS

A first small step to analysing the data would be to look at the "centroids" for each class, that is calculating the means for each class.

```
# calculate means
means <- array(dim = c(3, 2))
for (class_no in 1:3){
    means[class_no, 1] <- mean (x_app[classe_app == class_no, 1])
    means[class_no, 2] <- mean (x_app[classe_app == class_no, 2])
}
# plot the means for train data
plot (x_app, col=couleur_app, main='Means for train data', xlab='First feature',
    ylab='Second feature')
points (means, col=class_colors, p=8, cex=4)
```

LISTING 2.3
Calculating means

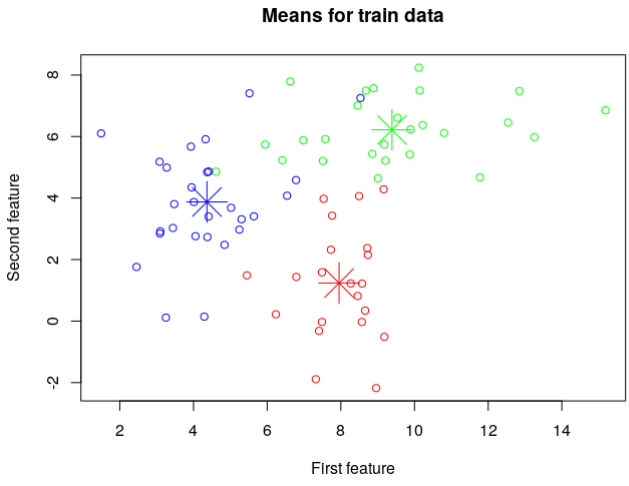

FIGURE 2.2
Visualising means with instances

True results

	Red	Green	Blue
X1	8	10	4
X2	1	6	4

My results

	Red	Green	Blue
X1	7.951986	9.389311	4.365176
X2	1.237545	6.218550	3.874318

The values[1] aren't so distant from the ground truth, so we can consider them insightful.

2.2.2 COVARIANCE MATRIX

Up next, we will calculate the covariance matrix. According to Wikipedia 2020a, covariance is a measure of how one feature changes in relation to another. Depending on the algorithm we're using, if two features are *strongly* correlated, it might be better to let one go, in order to reduce the *dimensionality* of the problem and make algorithms faster. But, for example, algorithms like Naive Bayes might actually benefit from correlated features. In our case, we only have 2 features and 75 instances, so the dimensionality of the problem doesn't really pose a problem for us.

```
sigma <- c()
for (class_no in 1:3){
    # number of instances = n_app = dim(x_app)[1]
    # number of features = dim(x_app)[2]
    sigma[[class_no]] <- matrix(1, dim(x_app)[2], dim(x_app)[2])
    for (i in 1:dim(x_app)[2])
    for (j in 1:dim(x_app)[2])
        sigma[[class_no]][i, j] <- cov(as.vector(x_app[classe_app == class_no, i]),
    as.vector (x_app[classe_app == class_no, j]))
}
```

LISTING 2.4
Calculating covariance

Below, we can find the script's result. For each class, we have calculated the *covariance matrix*. On the diagonal of each Σ_i matrix, we have the *variance* of each feature, and any other element $\Sigma_(j,k), j \neq k$, represents the correlation between the features j, k. Of course, these matrixes are symmetric.

$$\Sigma_1 = \begin{pmatrix} 0.95337723 & 0.09544548 \\ 0.09544548 & 3.26152120 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 5.8526867 & 0.6107413 \\ 0.6107413 & 1.0474417 \end{pmatrix},$$

$$\Sigma_3 = \begin{pmatrix} 2.0251769 & 0.7214545 \\ 0.7214545 & 3.0809579 \end{pmatrix}$$

$$\sigma_1 = \begin{pmatrix} 1 & 2 \end{pmatrix}, \sigma_2 = \begin{pmatrix} 3 & 1 \end{pmatrix} \sigma_3 = \begin{pmatrix} 1.5 & 2 \end{pmatrix}$$

By Σ_i we denoted the covariance matrix, and by σ_i the ground truth regarding standard deviation.

2.2.3 ANALYSING DIAGONAL VALUES

Looking at the values on the diagonal and comparing them (! by not forgetting to raise the standard deviation, that is values from σ_i, to the power of 2, so we get variance), to the true values, we can see that overall they are pretty close. The only exception is the variance for the first feature, for the second class; those aren't very similar (≈ 5.8 vs. 3^2). To cite Wikipedia 2020b, variance "measures how far a set of (random) numbers are spread out from their average value". That being said, it gives us a sense of dispersity in our data.

[1]The means were calculating only on the training dataset, not on all of the instances

Now let's apply the above observations to our data[2]. Before this, please note that in our plots above 2.1, the first feature X_1 was projected on the abscissa and the second feature X_2 on the ordinate. Looking at the red class and variance for the first atribute, we could that, for the first feature, the variance is not that high. That would mean that the red-class instances, projected on the x-axis, would be "gathered" around. If we look at the image, that is, indeed, true!

Thinking the other way around, if we look at the figure2.1 at X_1 for the green class, we can notice that the projections on the x-axis for the instances are quite dispersed. That would mean that the variance for that case is higher. And that is, once again, true (variance of ≈ 5.8 in our results), proof that our algorithm makes the cut.

2.2.4 ANALYSING CORRELATION BETWEEN FEATURES

Let us now note $\Sigma_i(j,k) = x$, a correlation between two features. Before analysing the values for x, we must take into account the following observations: if x is positive, then if the feature j increases, then so does the value for k. If x is negative, then if feature j increases, then the value for k decreases. If x is 0, then the features are not correlated. We say that two features j and k are strongly correlated when the x is large.

The covariance matrix tells us that for the train instances in the red class, there's almost no correlation, since the value is fairly low, ≈ 0.1. Even for the other classes, the correlation is still < 1, so we can conclude that the features are not highly correlated.

2.3 LINEAR DISCRIMINANT ANALYSIS

2.3.1 CONSTRUCTING THE LDA MODEL

We will now try to use *LDA* to find a linear combination between X_1 and X_2. This algorithm will also allow us to construct *boundaries* between classes. We'll illustrate this as a Voronoi diagram.

```
# construct the grid
xp1 <- seq(min(x_app[,1]), max(x_app[,1]), length=50)
xp2 <- seq(min(x_app[,2]), max(x_app[,2]), length=50)
grille <- expand.grid(x1=xp1, x2=xp2)
x_app.lda <- lda(x_app, classe_app)
grille <- cbind(grille[,1], grille[,2])
Zp <- predict(x_app.lda, grille)
# plot boundaries
zp<-Zp$post[,3]-pmax(Zp$post[,2],Zp$post[,1])
contour(xp1,xp2,matrix(zp,50),levels=0,drawlabels=FALSE)
zp<-Zp$post[,2]-pmax(Zp$post[,3],Zp$post[,1])
contour(xp1,xp2,matrix(zp,50),add=TRUE,levels=0,drawlabels=FALSE)
points(x_app, col=couleur_app, p=19)
```

LISTING 2.5
Running LDA on train data

[2]The covariance matrixes were calculated on the training data, so the observations we made are for that dataset

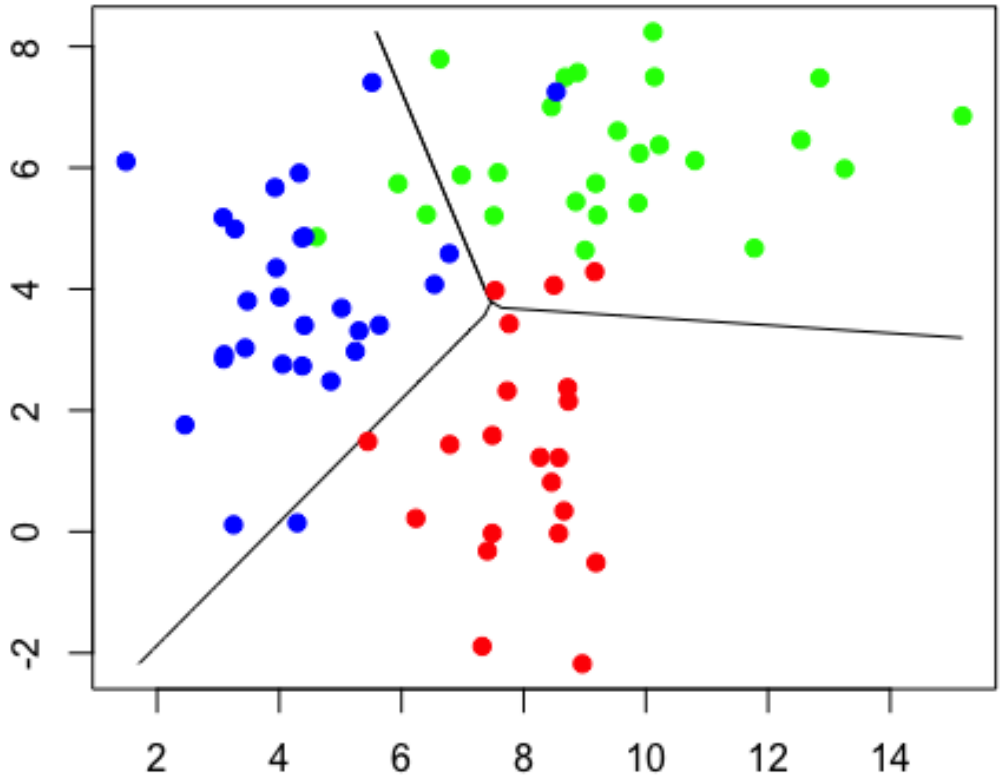

FIGURE 2.3
Decision lines on train dataset

The classification isn't perfect. For example, the line on the right, separating the green instances from the red ones, could have been higher, in order to include the 3 red instances that are classified as being green by our algorithm. But that would raise the question of *overfit*. So, overall, we can say that the algorithm *generalizes* the data well.

2.3.2 TESTING THE LINEAR COMINATION

Now that we have a linear combination between our features, we can see how well it behaves on our test data.

```
assigne_test<-predict(x_app.lda, newdata=x_test)
# Estimation des taux de bonnes classifications
table_classification_test <-table(classe_test, assigne_test$class)
# table of correct class vs. classification
diag(prop.table(table_classification_test, 1))
# total percent correct
taux_bonne_classif_test <-sum(diag(prop.table(table_classification_test)))
```

LISTING 2.6
Predicting with LDA

The script above tells us that we have an overall accuracy of 88%: 85% for the red class, 95% for the green class and $\approx 86\%$ for the blue class. If we are to plot the test data using the same boundaries as above, it would look like so:

```
forms <- 1:n_test
possible_forms <- c(15, 17, 19)
for (i in 1:n_test)
    forms[i] = possible_forms[assigne_test$class[i]]
# replot the grid & decision lines
xp1 <- seq(min(x_test[,1]), max(x_test[,1]), length=50)
xp2 <- seq(min(x_test[,2]), max(x_test[,2]), length=50)
grille <- expand.grid(x1=xp1,x2=xp2)
plot (grille)
zp<-Zp$post[,3]-pmax(Zp$post[,2],Zp$post[,1])
contour(xp1,xp2,matrix(zp,50),levels=0,drawlabels=FALSE)
zp<-Zp$post[,2]-pmax(Zp$post[,3],Zp$post[,1])
contour(xp1,xp2,matrix(zp,50),add=TRUE,levels=0,drawlabels=FALSE)
points(x_test, col=couleur_test, p=forms)
```

LISTING 2.7
Plotting predictions and true classes

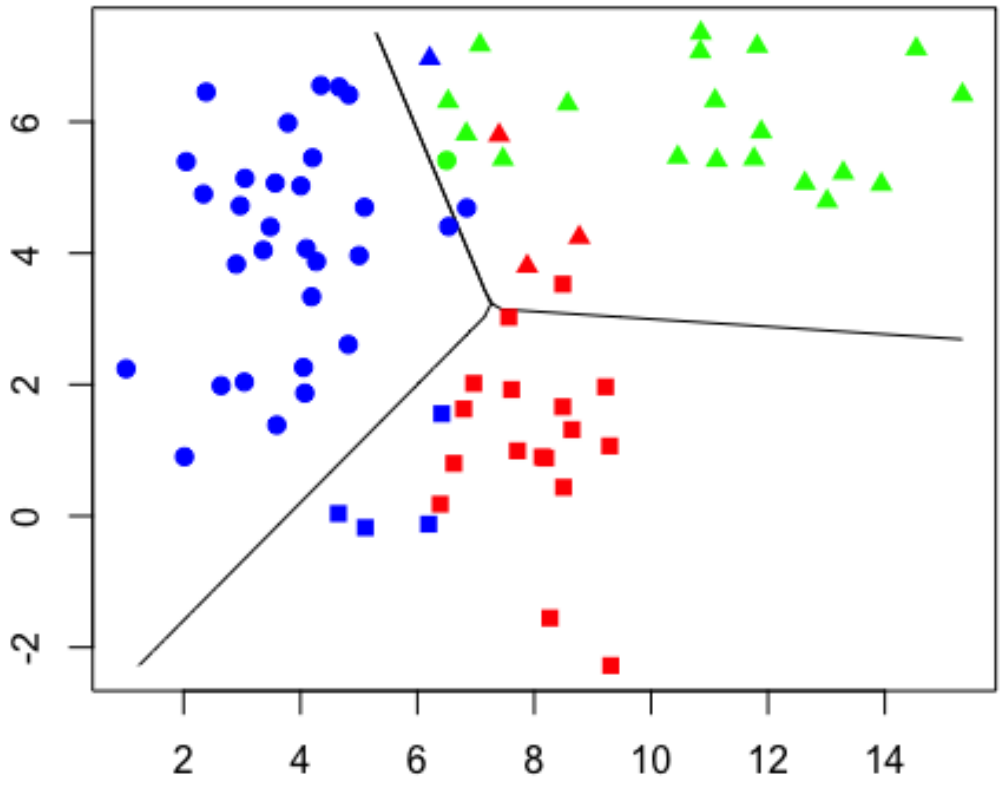

FIGURE 2.4
Predictions using LDA

In the figure above, the colors represent the true class of the instances, and each shape represents one of our predictions. For example, missclassifications are: blue instances that are not circles, red instances that are not squares and green instances that aren't triangles.

Counting the missclassifications mentioned above, we find a total of 9: 3 red, 1 green and 5 blue. We have a number of 75 instances in total: 20 red, 20 green and 35 blue. That gives as a total error rate of $9/75 = 12\%$. For each individual class, the errors are: $3/20 = 15\%$, $1/20 = 5\%$ and $5/35 \approx 14\%$. That is indeed what our algorithm gave us as well2.3.2.

2.4 QUADRATIC DISCRIMINANT ANALYSIS

The *Quadratic Classifier* brings, as an advantage over LDA, more generality. It still does a linear combination of the features, a feature's value may now be raised to a certain power: $f(x) = x^T * A * x + b^T x + c$. This will give us decision lines that will be more complex, since introducing terms like x^T in the equation can help us achieve curves for the boundaries.

To use QDA instead of LDA, we simply replace "lda" with "qda" in the code presented above. Here's how the results look:

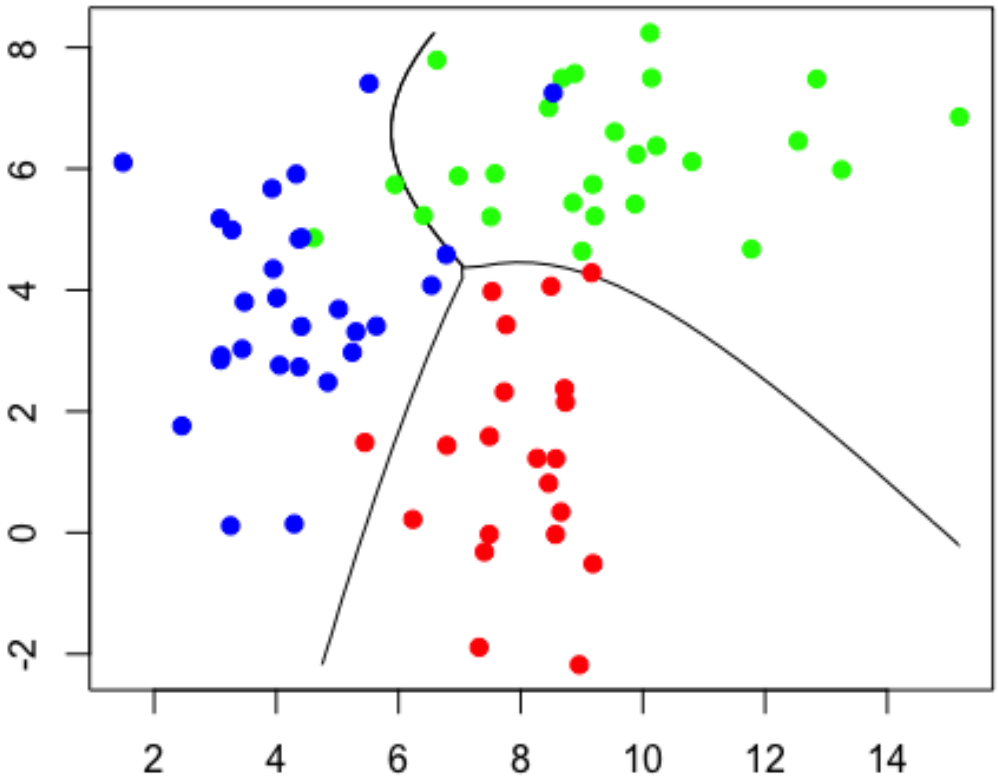

FIGURE 2.5
QDA trained on train dataset

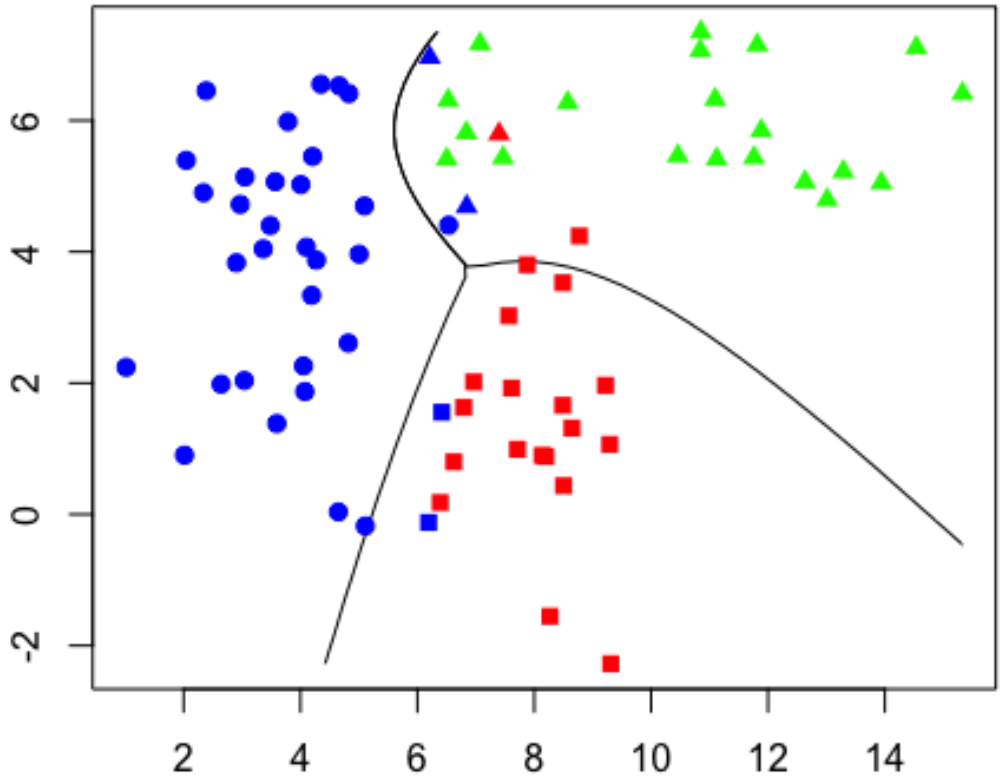

FIGURE 2.6
QDA trained on the test dataset

This time, we have a total accuracy of 93%: 95% for the red instances, 100% for the green ones and $\approx 86\%$ for the blue instances. Quadratic discriminant analysis is more complex, but at the same time this gives more generalization power to the algorithm. As we can see, the results using QDA are better than those from LDA.

CHAPTER 3

CONCLUSION

Our initial goal was to draw conclusion about points, about their x and y values and to classify them as well as possible, having as comparison the true labels. By using the covariance, LDA and QDA, we managed to do just that. We saw that LDA does a good job classifying the points, overall, but QDA introduced a notion of generality (the curved decision lines) that increased our algorithm's accuracy.

BIBLIOGRAPHY

Wikipedia (2020a). *Covariance*. Wikimedia Foundation. URL: https://en.wikipedia.org/wiki/Covariance (visited on 11th Mar. 2020).

– (2020b). *Variance*. Wikimedia Foundation. URL: https://en.wikipedia.org/wiki/Variance (visited on 11th Mar. 2020).