Linear/Quadratic Discriminant Analysis

Introduction

Let's compute some LDA/QDA classifiers and compare with the other classifiers we've worked with.

Set up

First, load the libraries.

```r
library(tidyverse)
```

```
## -- Attaching packages --------- tidyverse 1.2.1 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.3.0

## -- Conflicts ------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(MASS) ## for lda
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

And some data…

```r
data.df3 <- read.csv("LinearClassData3.csv")
data.df4 <- read.csv("LinearClassData4.csv")
```

To start use, data.df3

```r
data.df <-  data.df4
```

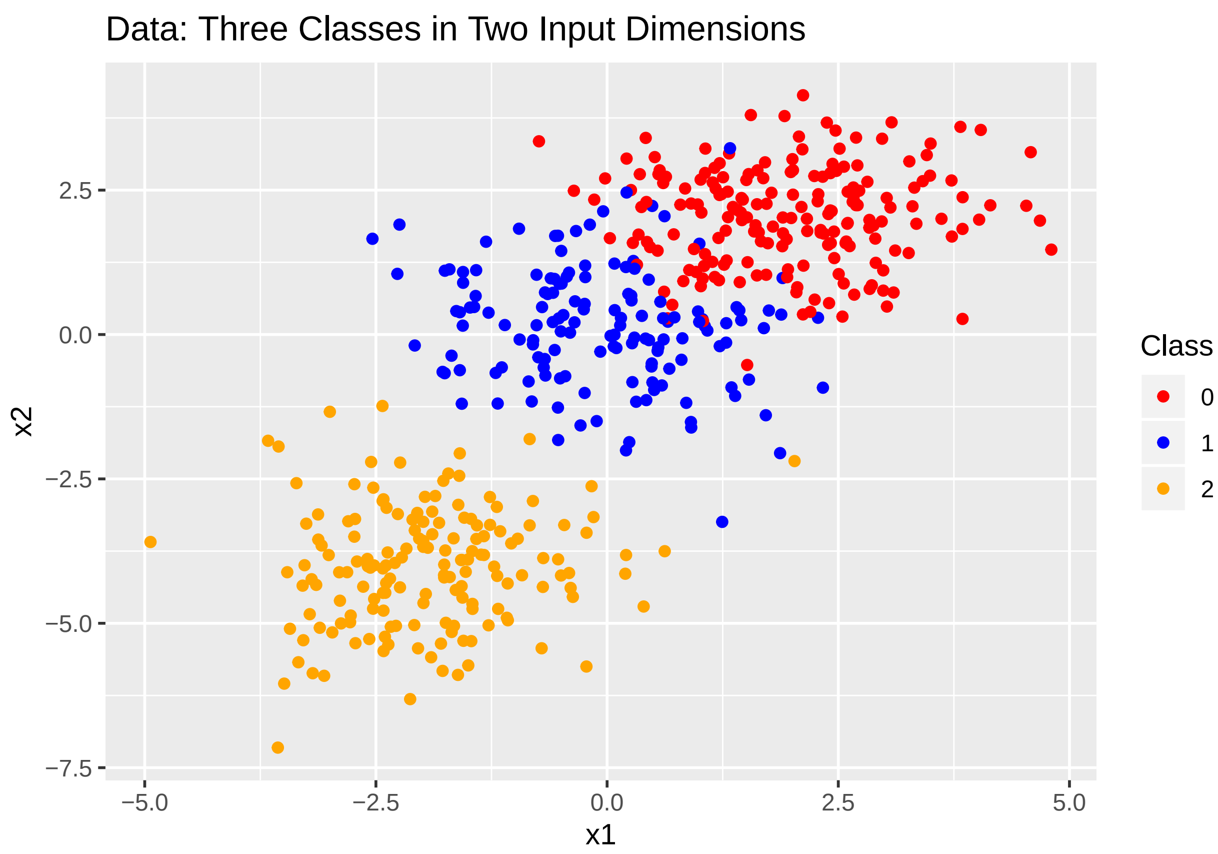
Look at the class distribution

```r
with(data.df,table(class))
```

```
## class
##   0   1   2
## 194 145 161
```

How's it look?

```
cols <- c("red","blue","orange")
data.df %>%
    ggplot() +
    geom_point(aes(x1,x2,color=factor(class)))+
    scale_color_manual(values=cols)+
  labs(title="Data: Three Classes in Two Input Dimensions",
       color="Class")
```



We have three classes, reasonably separated. ## Model Building Now let's build some models and classify.

One Hot Encoding

As a start, let's quickly use the One Hot Encoding to perform a classification,

```
data.df <- data.df %>%
    mutate(Y0=1*(class==0),
           Y1=1*(class==1),
           Y2=1*(class==2))
```

Build the three One Hot Models

```
mod0 <- lm(Y0 ~ x1 + x2, data=data.df)
mod1 <- lm(Y1 ~ x1 + x2, data=data.df)
mod2 <- lm(Y2 ~ x1 + x2, data=data.df)
```

And predictions from these models.

```
data.df$pred0 <- predict(mod0)
data.df$pred1 <- predict(mod1)
data.df$pred2 <- predict(mod2)
```

Add the predictions to the data frame.

```
data.df <- data.df %>%
    rowwise() %>%
    mutate(class.1hot=which.max(c(pred0,pred1,pred2))-1)
```

How does this look…look at confusion matrix and error rate

```
with(data.df,table(class,class.1hot))
```

```
##        class.1hot
## class   0   1   2
##     0 191   3   0
##     1  66  60  19
##     2   1   1 159
```

```
(err.hot <- with(data.df,mean(class != class.1hot)))
```

```
## [1] 0.18
```

Ok…

LDA: By Hand!

Let's build the LDA directly from the definitions. This means we need to compute the p=2 versions of the means and covariances.

Let's list all the variables we need to etimate for each $k = 0, 1, 2, \ldots C - 1$

- n: the number of observations.
- n_k: the class size, i.e. number of observations in each class (number)
- π_k: the class proportion (number)
- μ_k: the class mean of the observations, equivalently, the centroid of the inputs in each class (vector in $px1$ vector).
- Σ_k: The class covariance. (pxp matrix).

For LDA, we assume that all the covariances are identical, i.e $\Sigma_k = \Sigma$. As estimate of Σ is given by

$$\Sigma = \frac{1}{n - C} \sum_{k=0}^{C-1} (n_k - 1)\Sigma_k.$$

Given these values, the discriminant function for the k^{th} class is

$$\delta_k(x) = -\frac{1}{2}(x - \mu_k)^T \Sigma^{-1}(x - \mu) + \log \pi_k$$

where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.

As it turns out, this is a linear function of x.

This means computing means and covariances etc

Start with the overall size of the data frame.

```
N <- nrow(data.df)
```

Go class by class, compute proportions, means and scatter (= scaled covariance) matrices

```
data.df0 <- data.df %>%
    filter(class==0)
data.df1 <- data.df %>%
    filter(class==1)
data.df2 <- data.df %>%
    filter(class==2)
```

Get data from the classes

Class 0: compute n_0, π_0, μ_0, Σ_0.

Note:
* The use of the R command "data.matrix" to extract a matrix of the data. * The R function "cov" will give you the covariance matrix Σ_0.

```
mat0 <- data.matrix(data.df0[c("x1","x2")])
mu0 <- apply(mat0,2,mean)
n0 <- nrow(mat0)
pi0 <- n0/N
sigma0 <- cov(mat0)
```

Do the same for Class 1.

```
mat1 <- data.matrix(data.df1[c("x1","x2")])
mu1 <- apply(mat1,2,mean)
n1 <- nrow(mat1)
pi1 <- n1/N
sigma1 <- cov(mat1)
```

And for Class 2.

```
mat2 <- data.matrix(data.df2[c("x1","x2")])
mu2 <- apply(mat2,2,mean)
n2 <- nrow(mat2)
pi2 <- n2/N
sigma2<- cov(mat2)
```

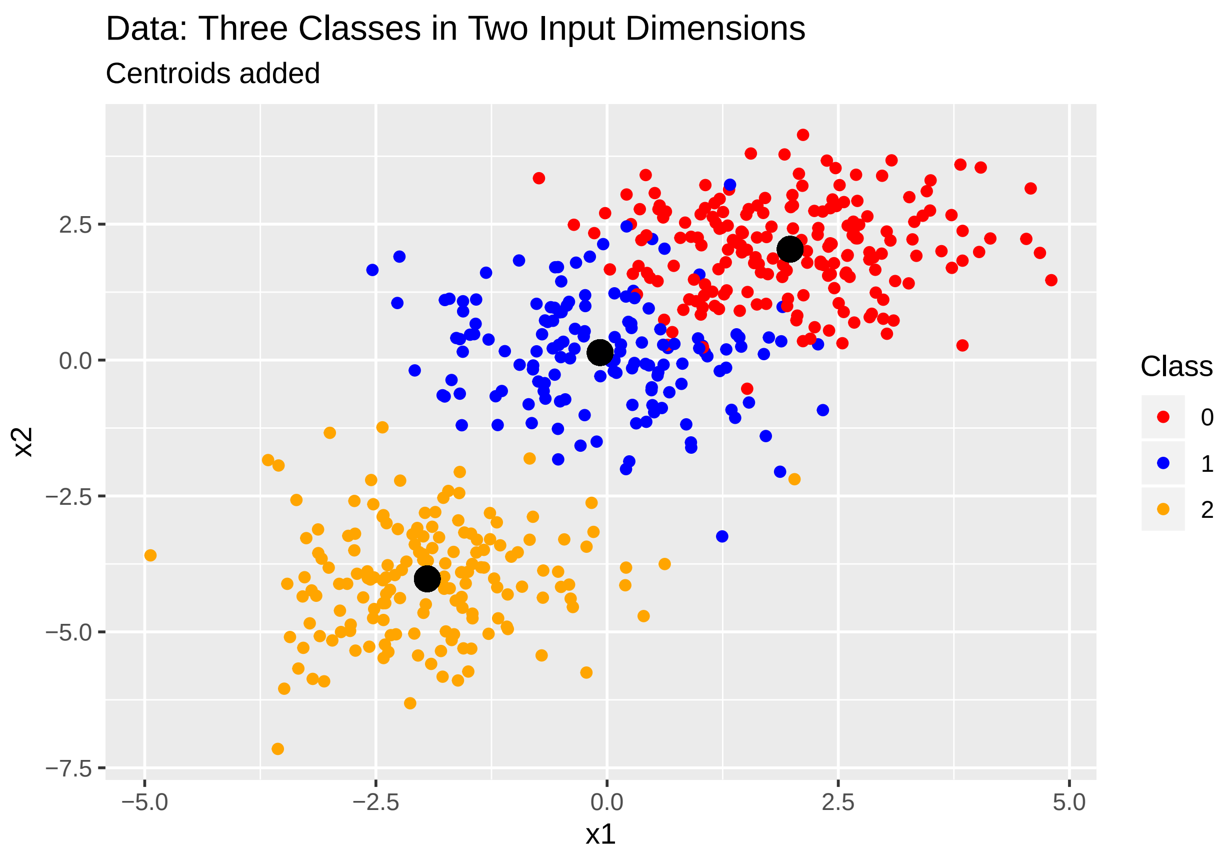
Check the class proportions.

```
c(pi0,pi1,pi2)
```

```
## [1] 0.388 0.290 0.322
```

Here it is worth taking a look at the means to see if there are correct. A quick ggplot will do this.

```r
data.df %>%
    ggplot() +
    geom_point(aes(x1,x2,color=factor(class)))+
    scale_color_manual(values=cols)+
    geom_point(aes(x=mu0[1],y=mu0[2]),color="black",size=4)+
    geom_point(aes(x=mu1[1],y=mu1[2]),color="black",size=4)+
    geom_point(aes(x=mu2[1],y=mu2[2]),color="black",size=4)+
  labs(title="Data: Three Classes in Two Input Dimensions",
       subtitle="Centroids added",
       color="Class")
```



Looks ok.

Compute the overall covariance estimator, this is a 2x2 matrix.

```r
(sig <- 1/(N-3)*((n0-1)*sigma0+(n1-1)*sigma1+(n2-1)*sigma2))
```

```
##              x1           x2
## x1  1.051729012 -0.006124708
## x2 -0.006124708  0.931100327
```

Compute the inverse. Use the R command "solve"

```r
invsig <- solve(sig)
```

It never hurts to check that the inverse is doing what it's supposed to do,

$$\Sigma\Sigma^{-1} = Id_p$$

where Id_p is the $p \times p$ identity matrix.
* Note "%*%" is matrix multiplication in R.

```
sig %*% invsig
```

```
##                x1 x2
## x1  1.000000e+00  0
## x2 -8.673617e-19  1
```

OK.. We're looking good. We can now compute the three discriminant functions. Directly implement the formulas for $\delta_k(x)$ as defined above.

```
discrim0 <- function(x){
    as.numeric(-1/2*t(x-mu0) %*% invsig %*% (x-mu0) +  log(pi0))
}
discrim1 <- function(x){
    as.numeric(-1/2*t(x-mu1) %*% invsig %*% (x-mu1) + log(pi1))
}
discrim2 <- function(x){
    as.numeric(-1/2*t(x-mu2) %*% invsig %*% (x-mu2)  +  log(pi2))
}
```

Using the discriminant functions

Apply the discriminants to data set and make the predictions.

```
data.df <- data.df %>%
    rowwise() %>%
    mutate(q0=discrim0(c(x1,x2)),
           q1=discrim1(c(x1,x2)),
           q2=discrim2(c(x1,x2))) %>%
    mutate(class2=which.max(c(q0,q1,q2))-1)
```

How's it look?

```
with(data.df,table(class,class2))
```

```
##      class2
## class   0   1   2
##     0 184  10   0
##     1  10 134   1
##     2   0   3 158
```

And the error rate. . .

```r
(err.lda1 <- with(data.df,mean(class != class2)))
```

```
## [1] 0.048
```

How do we compare?

```r
c(err.hot,err.lda1)
```

```
## [1] 0.180 0.048
```

Pretty close. These are both linear methods so they probably do about the same thing.

LDA the R way

Let's now apply the R lda function.

```r
mod.lda <- lda(class ~ x1+x2,data=data.df)
pred.lda <- predict(mod.lda)
data.df$class.lda <- pred.lda$class
```

With any luck, the last two methods should be identical!

```r
with(data.df,table(class2,class.lda))
```

```
##       class.lda
## class2   0   1   2
##      0 194   0   0
##      1   0 147   0
##      2   0   0 159
```

Phew...

Visualizations

The One Hot and LDA methods yield similar results. How similar are the models?

A visualization in the plane will help show the comparison.

One more grid.

```r
rng1 <- with(data.df, range(x1))
rng2 <- with(data.df, range(x2))
## Grid Size
gridSize <- 200
##grid values in x1 and x2 directions
vals1 <- seq(rng1[1],rng1[2],length=gridSize)
vals2 <- seq(rng2[1],rng2[2],length=gridSize)
## Build the grid
grid.vals <- expand.grid(vals1,vals2)
##Pack together as a data frame
grid.df <- data.frame(x1=grid.vals[,1],
                      x2=grid.vals[,2])
```

Predicton on the grid.

```
pred.grid <- predict(mod.lda,newdata=grid.df)
grid.df$class.lda <- pred.grid$class
```

What are we looking at...

```
lda.gg <- grid.df %>%
  ggplot()+
  geom_tile(aes(x1,x2,fill=class.lda),alpha=.37)+
  geom_point(data=data.df,aes(x1,x2,color=factor(class)))+
  scale_color_manual(values=cols) +
  scale_fill_manual(values=cols)  +
  labs(title="LDA Model: Three Classes",
       color="Class")+
  guides(fill=F)
lda.gg
```

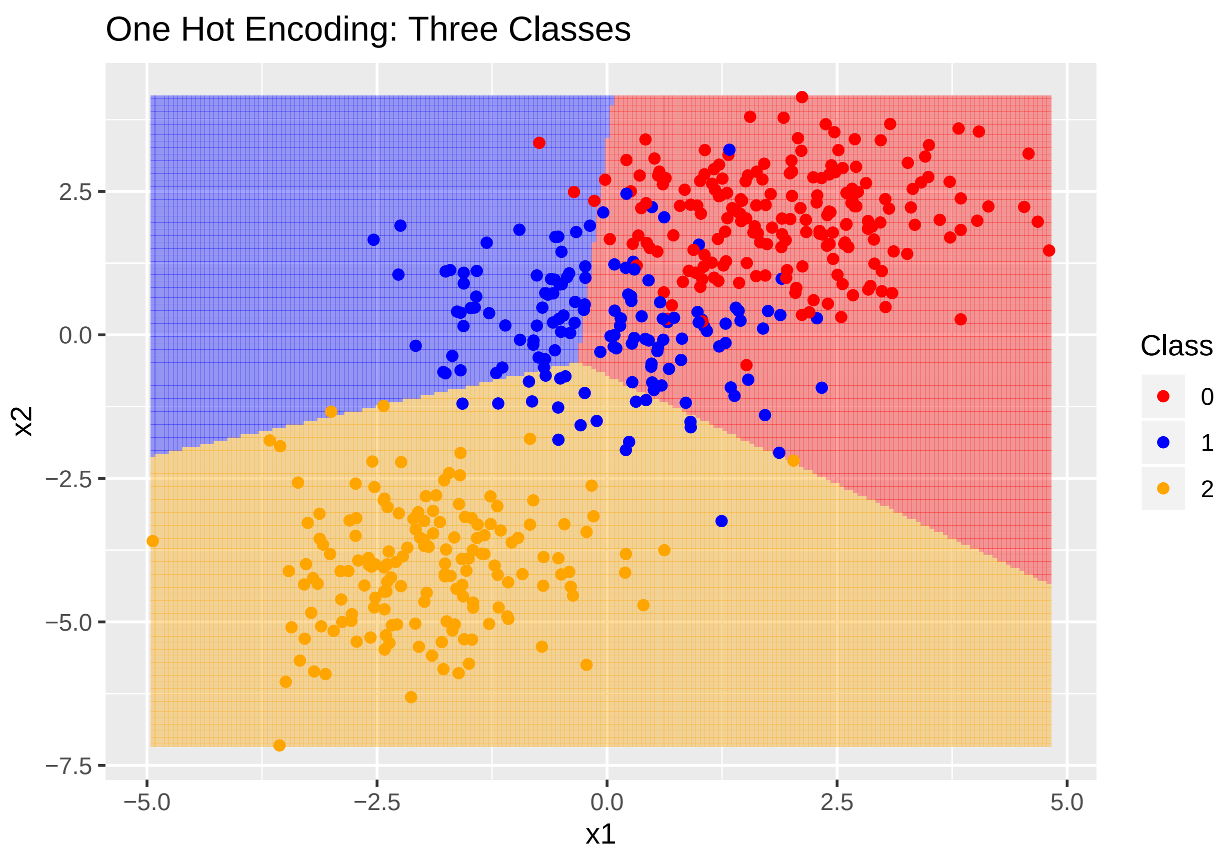
Now do the same for our One Hot Model.

Make the three separate predictions on the grid and then use these to make the class prediction.

```
p0 <- predict(mod0,newdata=grid.df)
p1 <- predict(mod1,newdata=grid.df)
p2 <- predict(mod2,newdata=grid.df)
grid.df$class.hot <- apply(cbind(p0,p1,p2),1,which.max)
```

What does this look like?

```
hot.gg <- grid.df %>%
  ggplot()+
  geom_tile(aes(x1,x2,fill=factor(class.hot)),alpha=.37)+
  geom_point(data=data.df,aes(x1,x2,color=factor(class)))+
  scale_color_manual(values=cols) +
  scale_fill_manual(values=cols)  +
  labs(title="One Hot Encoding: Three Classes",color="Class")+
  guides(fill=F)
hot.gg
```



Pretty darn similar.

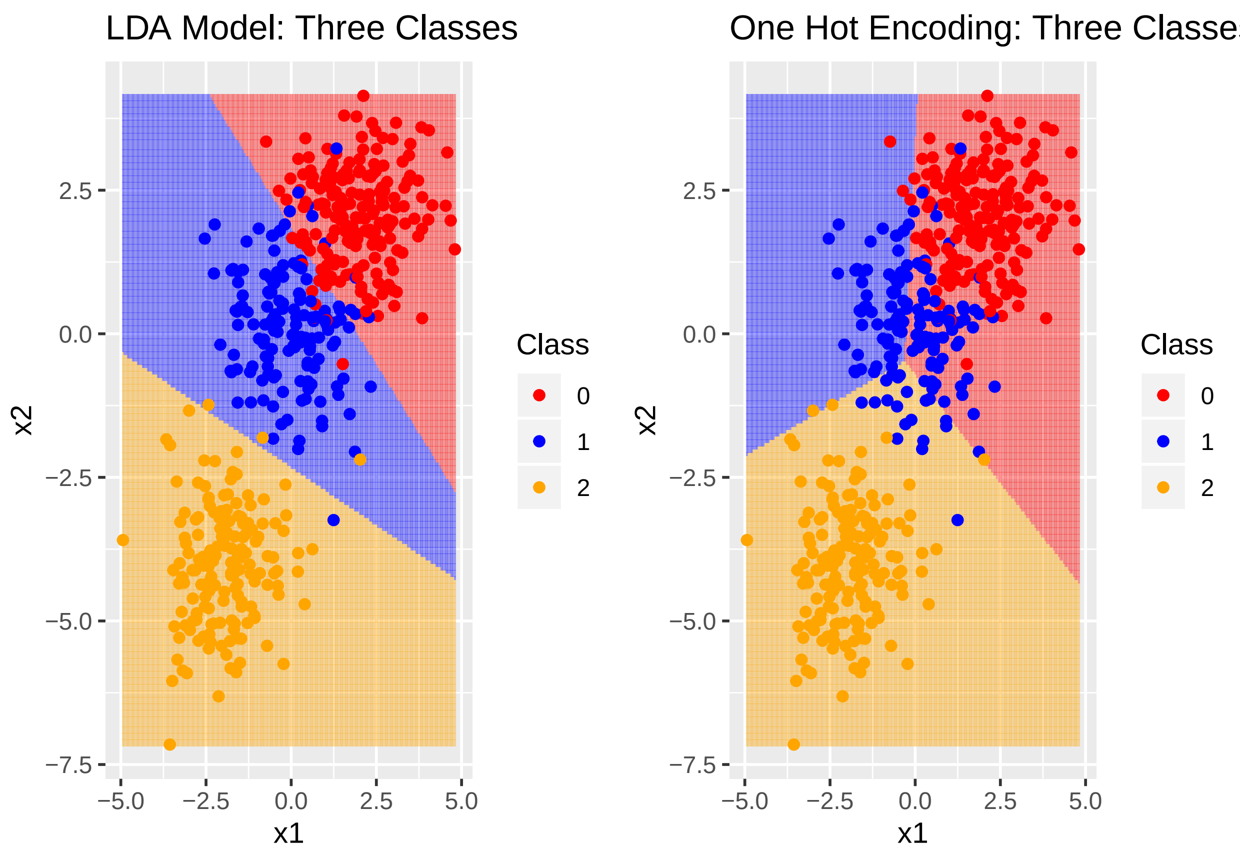
Load in gridExtra so we can use grid.arrange

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
grid.arrange(lda.gg,hot.gg,nrow=1)
```



There are some subtle differences.

Alternative Three Class Data set.

Quick computation: Repeat everything above with data.df4.

As the models and the errors indicate, in this case, LDA and One Hot perform very differently.

```r
c(err.lda1,err.hot)
```

```
## [1] 0.048 0.180
```

Quadratic Discriminant Analysis.

Both One Hot and LDA can be extended to include non-linear features.

One Hot with Interactions.

Modifiy the One Hot models to include all possible quadratic terms.

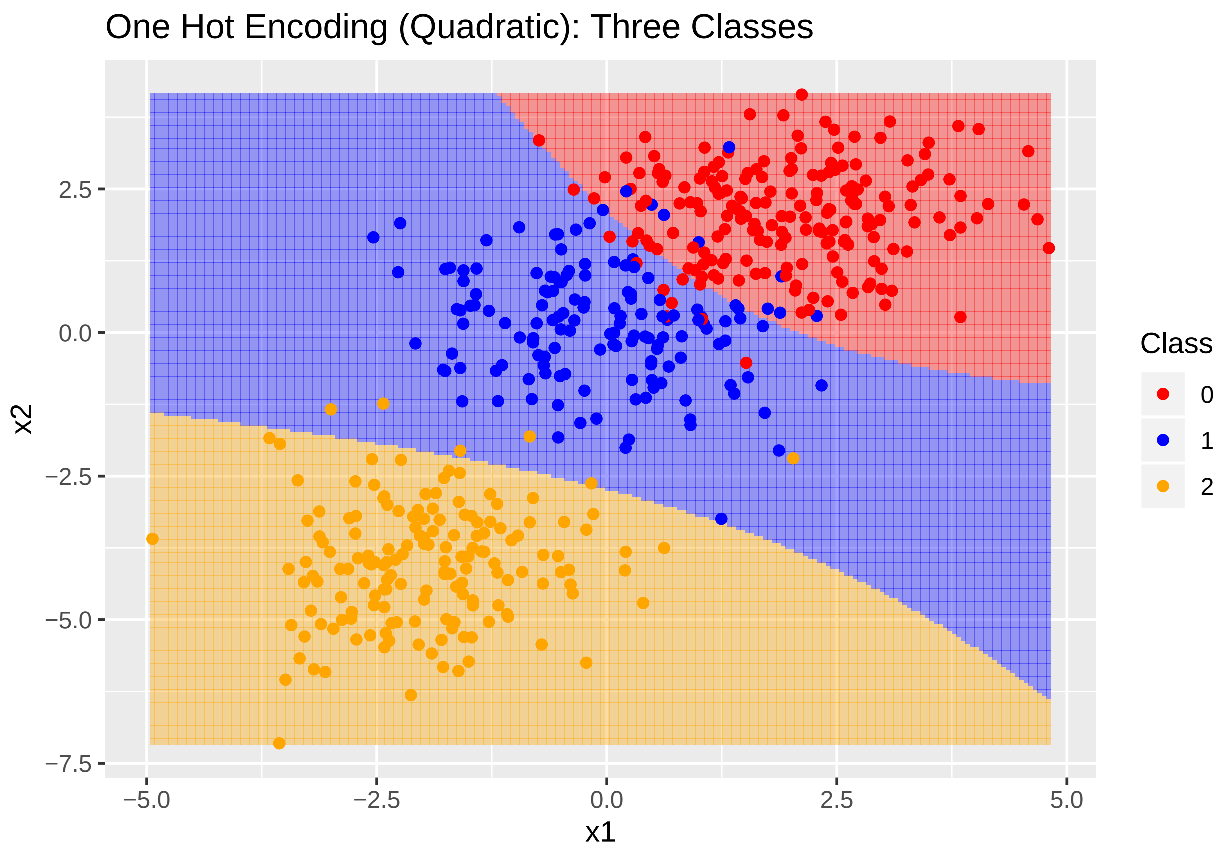
```r
mod0 <- lm(Y0 ~ x1 + x2 + I(x1*x2)+I(x1^2)+I(x2^2), data=data.df)
mod1 <- lm(Y1 ~ x1 + x2 + I(x1*x2)+I(x1^2)+I(x2^2), data=data.df)
mod2 <- lm(Y2 ~ x1 + x2 + I(x1*x2)+I(x1^2)+I(x2^2), data=data.df)
```

Now just add the predictions on the grid and proceed as before.

```
p0 <- predict(mod0,newdata=grid.df)
p1 <- predict(mod1,newdata=grid.df)
p2 <- predict(mod2,newdata=grid.df)
grid.df$class.hot2 <- apply(cbind(p0,p1,p2),1,which.max)
```

And the result…

```
hot2.gg <- grid.df %>%
  ggplot()+
  geom_tile(aes(x1,x2,fill=factor(class.hot2)),alpha=.37)+
  geom_point(data=data.df,aes(x1,x2,color=factor(class)))+
  scale_color_manual(values=cols) +
  scale_fill_manual(values=cols)   +
  labs(title="One Hot Encoding (Quadratic): Three Classes",color="Class")+
  guides(fill=F)
hot2.gg
```



Allowing quadratic terms in the model produces a much better fit.

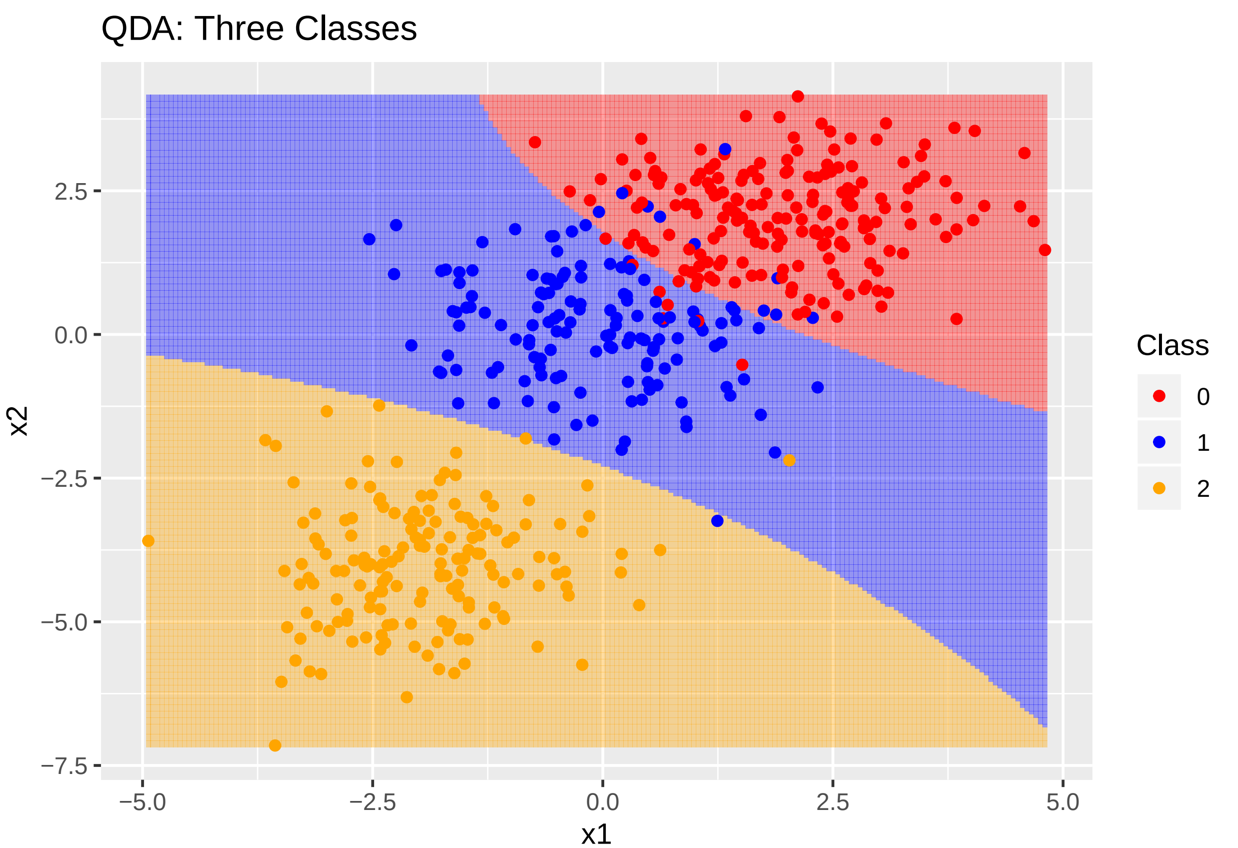
QDA

QDA also involves quadratic terms. Recall that is this case, we are allowing non-equal covariances in the classes.

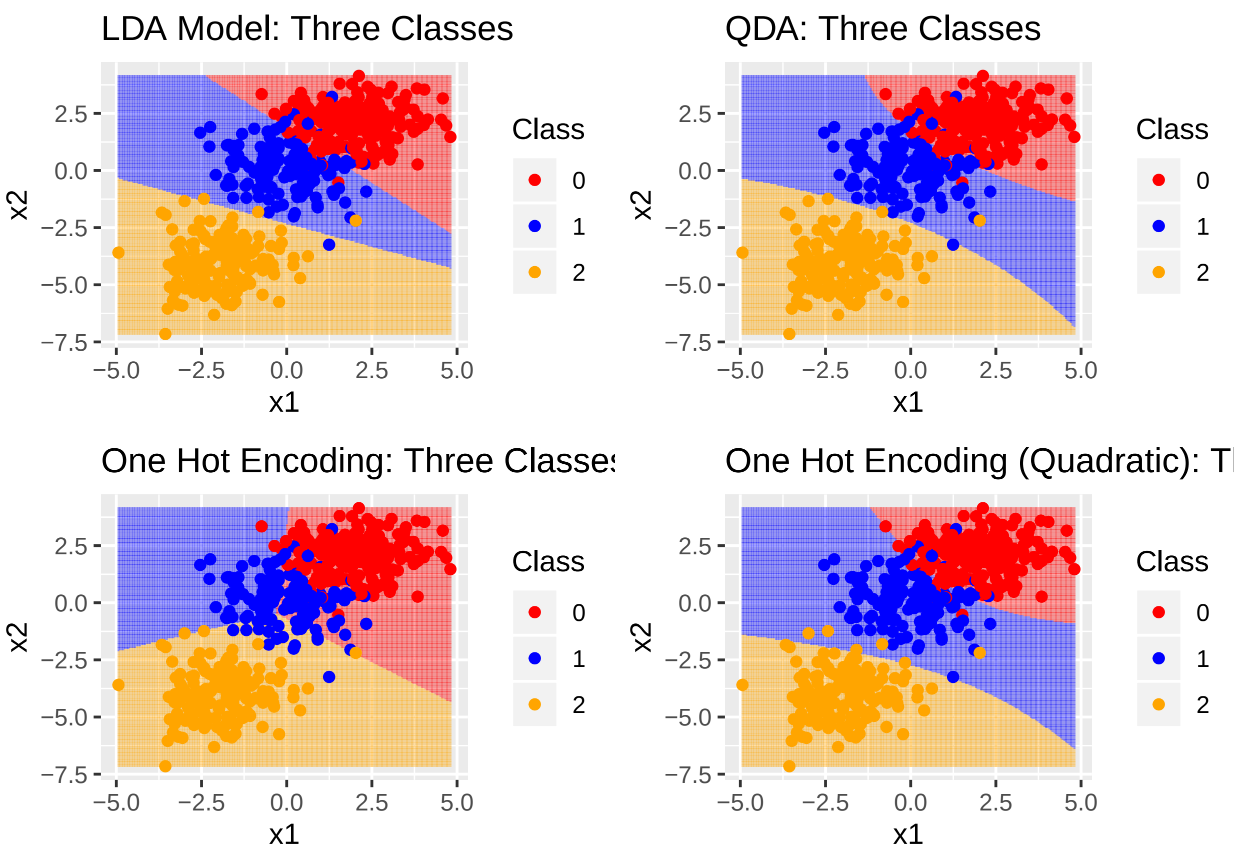
```
mod.qda <- qda(class ~ x1+x2,data=data.df)
pred.qda <- predict(mod.qda,newdata=grid.df)
grid.df$class.qda <- pred.qda$class
```

How’s this look?

```
qda.gg <- grid.df %>%
  ggplot()+
  geom_tile(aes(x1,x2,fill=class.qda),alpha=.37)+
  geom_point(data=data.df,aes(x1,x2,color=factor(class)))+
  scale_color_manual(values=cols) +
  scale_fill_manual(values=cols)   +
  labs(title="QDA: Three Classes",color="Class")+
  guides(fill=F)
qda.gg
```




```
grid.arrange(lda.gg,qda.gg,hot.gg,hot2.gg,nrow=2)
```



An interesting montage of plots.