LDA as a Dimension Reduction Method

First, load the libraries.

```
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()
```

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

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

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

Introduction

LDA does more than just estimate the Bayes Classifier. It also serves as a means of reducing the number of features (predictors) in a data set. This approach is most valuable when there are reasonably large number of predictors ($p > 4$ or so).

Here we will look at in the fairly simple case with $C = 3$ classes and $p = 2$ predictors.

FACT: LDA always projects data onto a the best (best separation) subspace of dimension $C - 1$.

Read the data

And some data...

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

To start use, data.df3

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

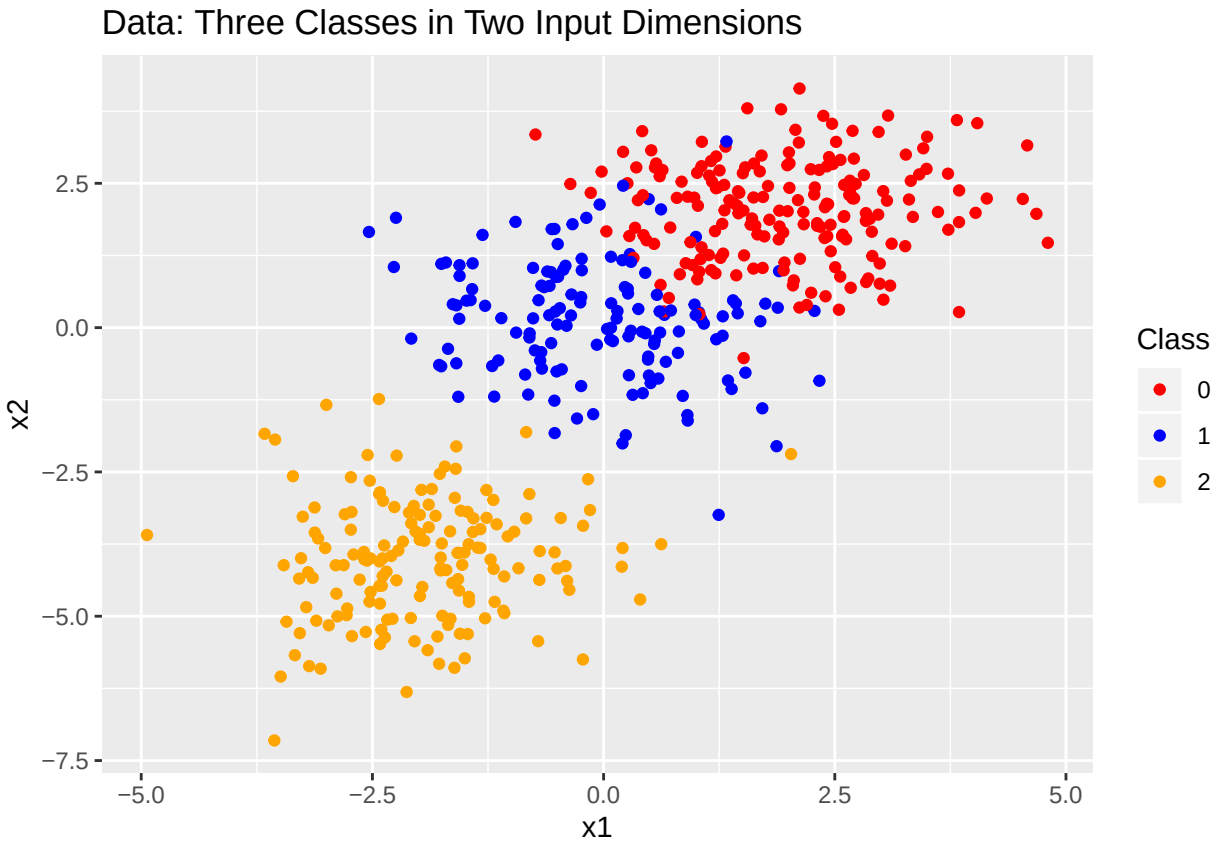
Look at the class distribution

```
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")
```



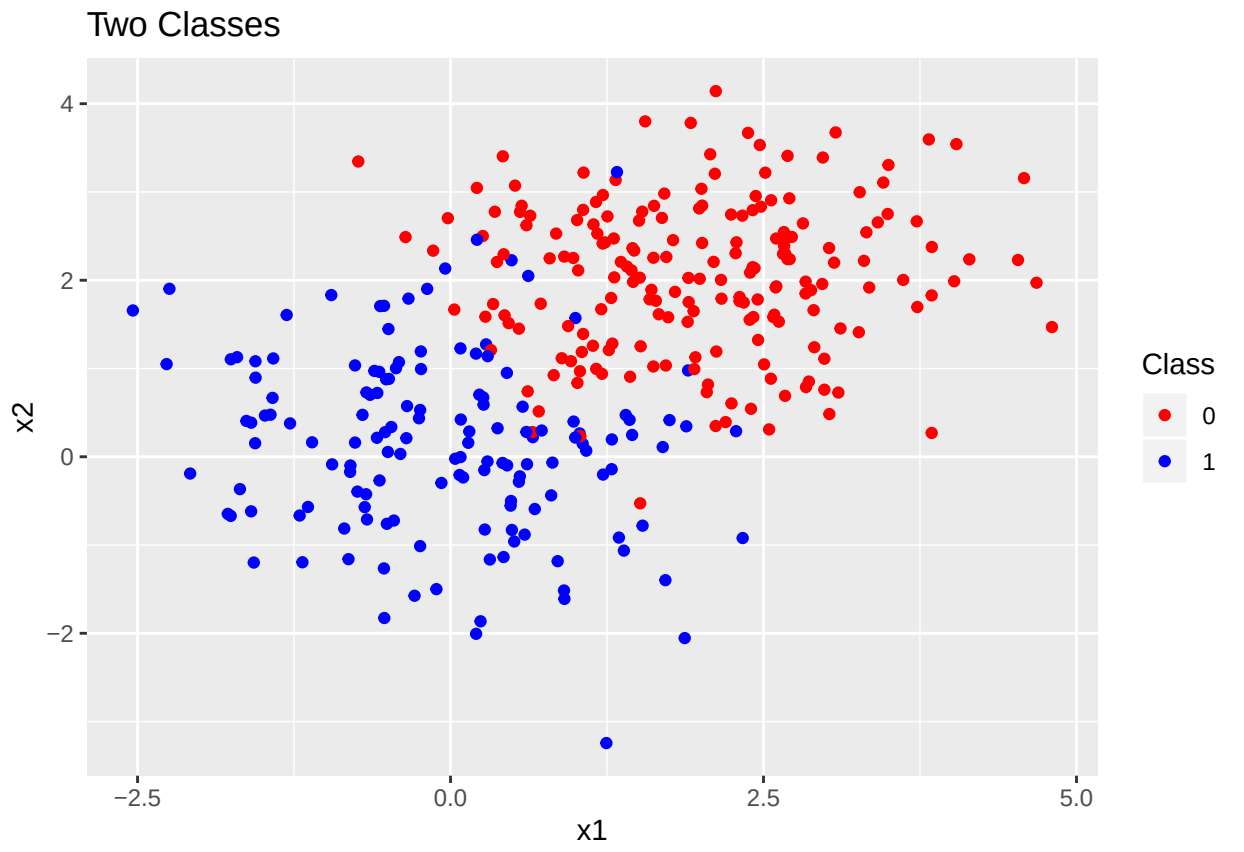
We have three classes, reasonably separated.

C=2 Data

Let's just use a subset of our existing data.

```
data.df <- data.df %>%
  filter(class != 2) %>%
  dplyr::select(x1,x2,class)
```

```
data.gg <- data.df %>%
  ggplot()+
  geom_point(aes(x1,x2,color=factor(class)))+
  scale_color_manual(values=cols)+
  labs(title="Two Classes",
       color="Class")
data.gg
```



Compute the LDA

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

Take a peek at the model. There's a lot of interesting stuff in there. Prior probabilities π_k, group means μ_k, and the "Coefficients of linear discriminants."

```
mod.lda
```

```
## Call:
## lda(class ~ x1 + x2, data = data.df)
##
```

```
## Prior probabilities of groups:
##         0         1
## 0.5722714 0.4277286
##
## Group means:
##            x1        x2
## 0  1.97944092 2.0397496
## 1 -0.07547554 0.1376003
##
## Coefficients of linear discriminants:
##           LD1
## x1 -0.6854702
## x2 -0.8006393
```

We are most interested in the last items, "Coefficients of linear discriminants".

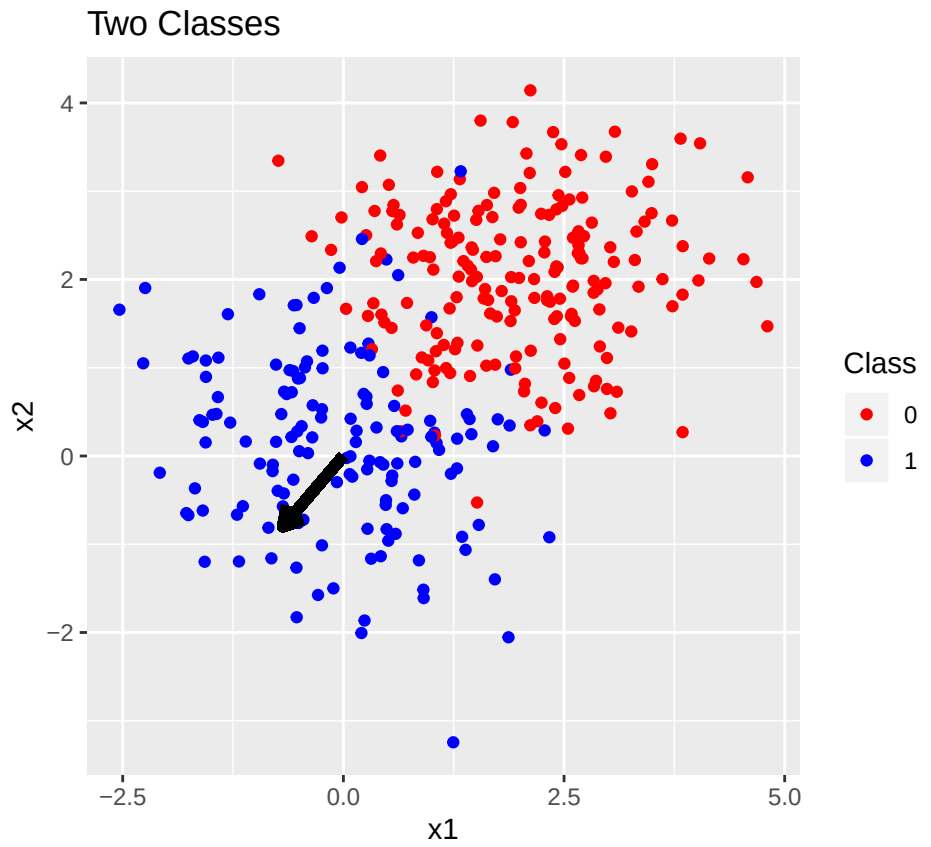
It's under the name "scaling". This is the direction of optimal projection!

```
(scale.lda <- mod.lda$scaling)
```

```
##           LD1
## x1 -0.6854702
## x2 -0.8006393
```

Add this to our ggplot as a vector. Note the "coord_fixed" which makes the aspect ratio=1 (same scale in each direction).

```
data.gg+
  geom_segment(aes(x=0,xend=scale.lda[1],
                   y=0,yend=scale.lda[2]),color="black",size=2,
                arrow = arrow(length = unit(0.03, "npc")))+
  coord_fixed()
```



The direction defined by this vector is the one-dimensional reduction of the data set. To achieve the reduction, we simply map each point onto the vector via orthogonal projection.

A little linear algebra

If $\mathbf{v}$ is a unit (length=1) vector and $\mathbf{x}$ is any other vector, the projection of $\mathbf{x}$ onto $\mathbf{v}$ is defined by

$$\text{proj}_{\mathbf{v}}\mathbf{x} = \alpha\mathbf{v}$$

where α is given by the dot product

$$\alpha = \mathbf{x} \cdot \mathbf{v}.$$

Simple Example

Suppose

$$\mathbf{v} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

Let's project the two vectors

$$\mathbf{x_1} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

and

$$\mathbf{x_2} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

```
v <- c(2,3)
x1 <- c(-1,1)
x2 <- c(5,1)
```

First, normalize $\mathbf{v}$ by dividing it by it's length.

Formally, If $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ is a vector in R^2. Then its length is simply

$$||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2}.$$

Normalizing a vector means rescaling it so the vector has length 1. In other words, replace $\mathbf{v}$ by

$$\frac{\mathbf{v}}{||\mathbf{v}||} = \frac{\mathbf{v}}{\sqrt{v_1^2 + v_2^2}}.$$

Let's do it (in R) to our vector

```
v <- v/sqrt(sum(v^2))
```

Dot productions in R are easy. Essentially, dot products and matrix multiplication are the same thing.

Note that "%*%" is matrix multiplication in R.

Here are the projections via dot products.

```
x1 %*% v
```

```
##           [,1]
## [1,] 0.2773501
```

```
x2 %*% v
```

```
##          [,1]
## [1,] 3.605551
```

To both of these calculations in one fell swoop, pack x1 and x2 into a matrix,

```
(x.mat <- matrix(c(x1,x2),byrow=T,ncol=2)) ##byrow makes sure we go across the row
```

```
##      [,1] [,2]
## [1,]   -1    1
## [2,]    5    1
```

We can get both the projections using matrix multiplication.

```
x.mat %*% v
```

```
##           [,1]
## [1,] 0.2773501
## [2,] 3.6055513
```

Back to our data.

Hence, we need to:

- Make the scaling vector have length 1 (easy)
- Project each data point on the scaling vector. The α values will be our one-dimensional reductions.
- To do the projections in "one fell swoop", create a matrix ($n \times 2$) of the input values. Use matrix multiplication to do all the inner products.

First, make scale.lda a unit vector

```
scale.lda <- scale.lda/sqrt(sum(scale.lda^2))
```

To do the all the dot products, use matrix multiplication.

Put all the input data into a matrix.

```
data.mat <- data.matrix(data.df[c("x1","x2")])
```

We can now compute all the projections of all of the data points in one fell swoop using matrix multiplication. Add these to the data frame as the projection values.

```
proj.vals <- data.mat %*% scale.lda ##n x 1 matrix
data.df$proj <- as.numeric(proj.vals)  #as. numeric is handy here.
```

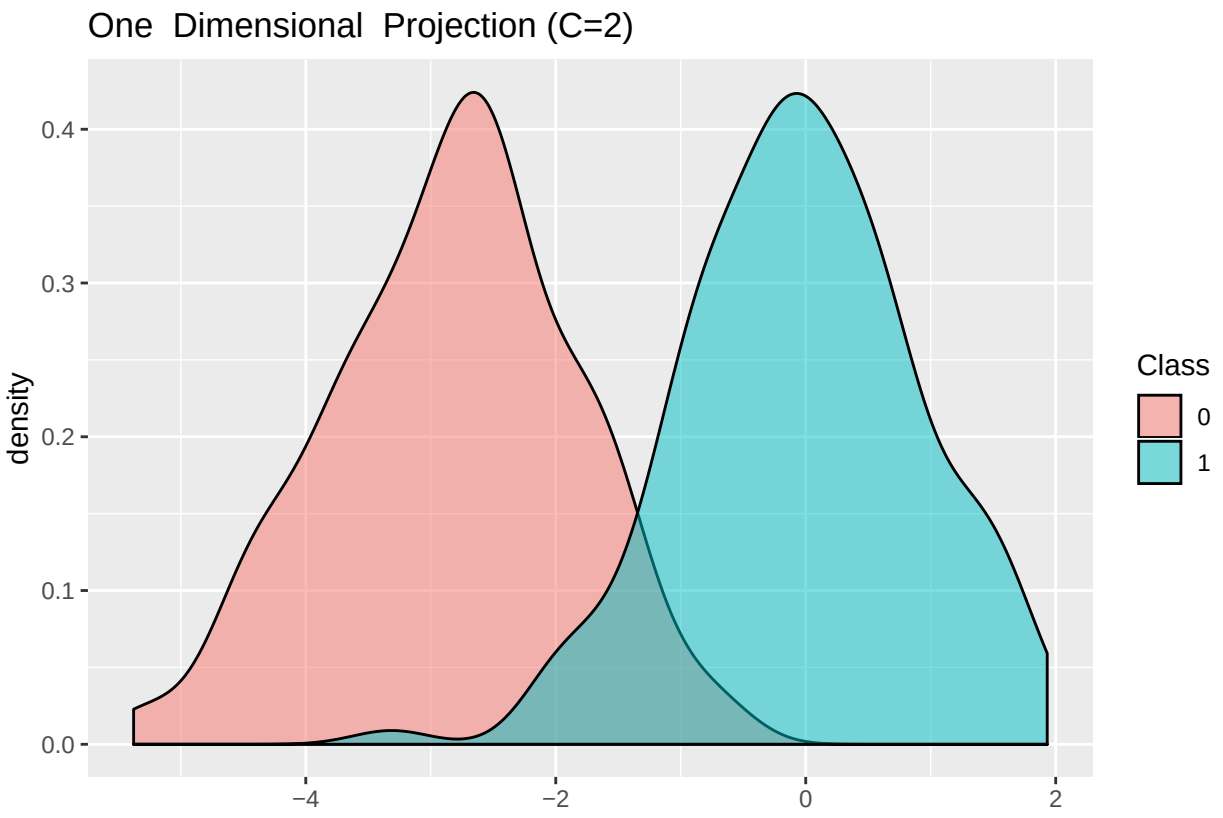
Let's look at what we've done.

```
data.df %>%
  ggplot()+
  ##Plot the points
  geom_point(aes(x1,x2,color=factor(class)))+
  ## Plot the projections
  geom_point(data=data.df,aes(x=scale.lda[1]*proj,
                              y=scale.lda[2]*proj,color=factor(class)),
             size=2)+
  ##the projection  lines,
  geom_segment(aes(x=scale.lda[1]*proj,xend=x1,
                   y=scale.lda[2]*proj,yend=x2,color=factor(class)),
               size=0.5,alpha=0.5)+
  ##the vector representing lda.scale
  geom_segment(aes(x=0,xend=scale.lda[1],
                   y=0,yend=scale.lda[2]),color="black",size=1.5,
                arrow = arrow(length = unit(0.03, "npc")))+
  ## Use these colors
  scale_color_manual(values=cols)+
  labs(title="Two Classes",
       subtitle="LDA Projection onto One Dimension",
       color="Class")+
  coord_fixed()
```

Nice picture!

It is also worthwhile to look at the resut in the reduced dimension space (in this case, one-dimensional).

```
data.df %>%
  ggplot()+
  geom_density(aes(proj,fill=factor(class)),alpha=.5)+
 labs(title="One  Dimensional  Projection (C=2)",
      fill="Class",
      x="")
```



Scoring a projection

The point of the LDA projection is that it maximizes the "Between Groups" and "With Groups" ratio. The score is a function of the direction of the projection line.

Call this direction $\mathbf{v}$.

Between Groups Scatter

The **Between Groups Scatter**, $\hat{S}_B$, is just the square of the difference of the means of the projected values. That is, if $\hat{\mu}_k$ = mean of the projections in class k, then

$$\hat{S}_B = (\mu_1 - \mu_0)^2.$$

Within Groups Scatter

In each group, define the **Within Groups Scatter**, $\hat{S}_k$ by

$$\hat{S}_k = \sum_{class\ k} (proj - \mu_k)^2$$

That is, the sum of the squares of the differences between the projections and the means. The sum is over the projections that belong to class k.

This is also the same as

$$\hat{S}_k = (n_k - 1)\text{Var}_k(proj)$$

Where $\text{Var}_k(proj)$ is the variance of the projectionsin class k. This is a handy way to think of it because R loves (absolutely) to calculate variances.

Given $\hat{S}_1$ and $\hat{S}_2$, the **Within Groups Scatter** $\hat{S}_W$ is defined as:

$$\hat{S}_W = \hat{S}_1 + \hat{S}_2.$$

Finally, the score of projection for a direction $J(\mathbf{v})$ is

$$J(\mathbf{v}) = \frac{\hat{S}_B}{\hat{S}_W}.$$

Assignment 1

Start with the $\mathbf{v}_{lda}$=lda.scaling direction and the already computed projection values. Compute $\hat{S}_B$ and $\hat{S}_W$ to show that

$$J(\mathbf{v}_{lda}) \approx 0.0255$$

This is should be the maximal score.

This score is important. Among all possible projection directions, LDA picks out the one that maximizes this value!

Assignment 2

It is true that among all possible choices of the projection direction, LDA picks out the direction of maximal score. Let's confirm this statement computationally.

Go back up to where the vector **scale.lda** is introduced into the projection process. From there, follow through all the way to the score value. Notice that you could change **scale.lda** to anything you want and the rest of the calculations would work fine.

Step one.

Redefine **scale.lda** to point in its perpendicular direction.

Hint: If $\mathbf{v} = (v_1, v_2)$ the a vector perpendicular to $\mathbf{v}$) is $(-v_2, v_1)$.

Reconstruct the sequence of steps with this new vector, $\mathbf{w}$, playing the role as the scaling vector. In particular, create the two plots (one showing the orthogonal projections and the other showing the densities of the projections in R^1) along with a value for

$$J(\mathbf{w}).$$

Step two.

Create a function that will compute

$$J(\mathbf{w})$$

for any **vector** w.

Note that since **scale.lda** is eventually normalized to have length=1. Hence, you only need to look at the direction of the vector $\mathbf{w}$. Vectors in R^2 of length one can be parameterized via

$$(\cos(\theta), \sin(\theta)).$$

Your task: Write a function called **scoreProj** that takes a single parameter - θ - and returns the score associated with projecting the data onto that direction

$$(\mathbf{w} = (\cos(\theta), \sin(\theta)).$$

scoreProj:
args: theta (a value between 0 and pi)
return: the score $J(\mathbf{v})$ where v is a unit vector in the θ direction, i.e. $\mathbf{v} = (\cos(\theta), \sin(\theta))$.

Demonstrate (graphically and numerically) that **scoreProj** is maximized at scale.lda's direction.

What is the direction that **minimizes** this **scoreProj**? Is it the direction perpendicular to the maximal direction?

Of course, your work should include any necessary and illuminating graphics. As with the threshold work last week, you will want to plot this function over a sequence of values between 0 and π.

You should end up with something like this to illustrate the value of the score as a function of θ