More Classification: ISL 4

DJM

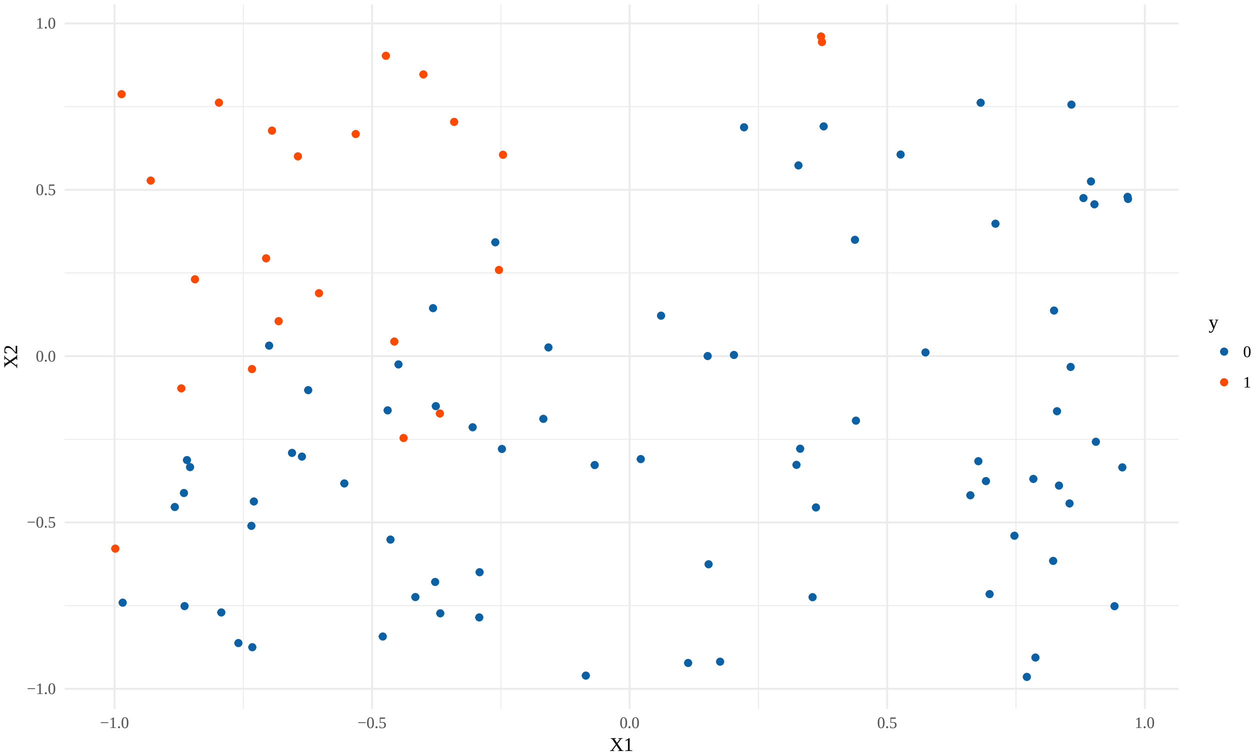
3 April 2018

Linear classifiers

Logistic regression

- Logistic regression is one example of a ~~linear classifier~~.
- It produces a line (or plane or hyperplane) which ~~separates~~ the two classes.

```
g <- ggplot(df, aes(X1,X2,color=y)) + geom_point() +
  scale_color_manual(values=c(blue,red))
g
```



```
log.lm = glm(y~.,data=df, family='binomial')
summary(log.lm)
```

```
##
## Call:
## glm(formula = y ~ ., family = "binomial", data = df)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.63909  -0.24256  -0.03246  -0.00061   2.35104
```

```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -3.803      1.010  -3.764 0.000167 ***
## X1            -6.593      1.918  -3.438 0.000586 ***
## X2             7.271      1.945   3.739 0.000185 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 107.855  on 99  degrees of freedom
## Residual deviance:  36.785  on 97  degrees of freedom
## AIC: 42.785
##
## Number of Fisher Scoring iterations: 8
```

What is the line?

- Suppose we decide "Predict `1` if `predict(log.lm) > 0.5`".
- This means "For which combinations of `x1` and `x2` is

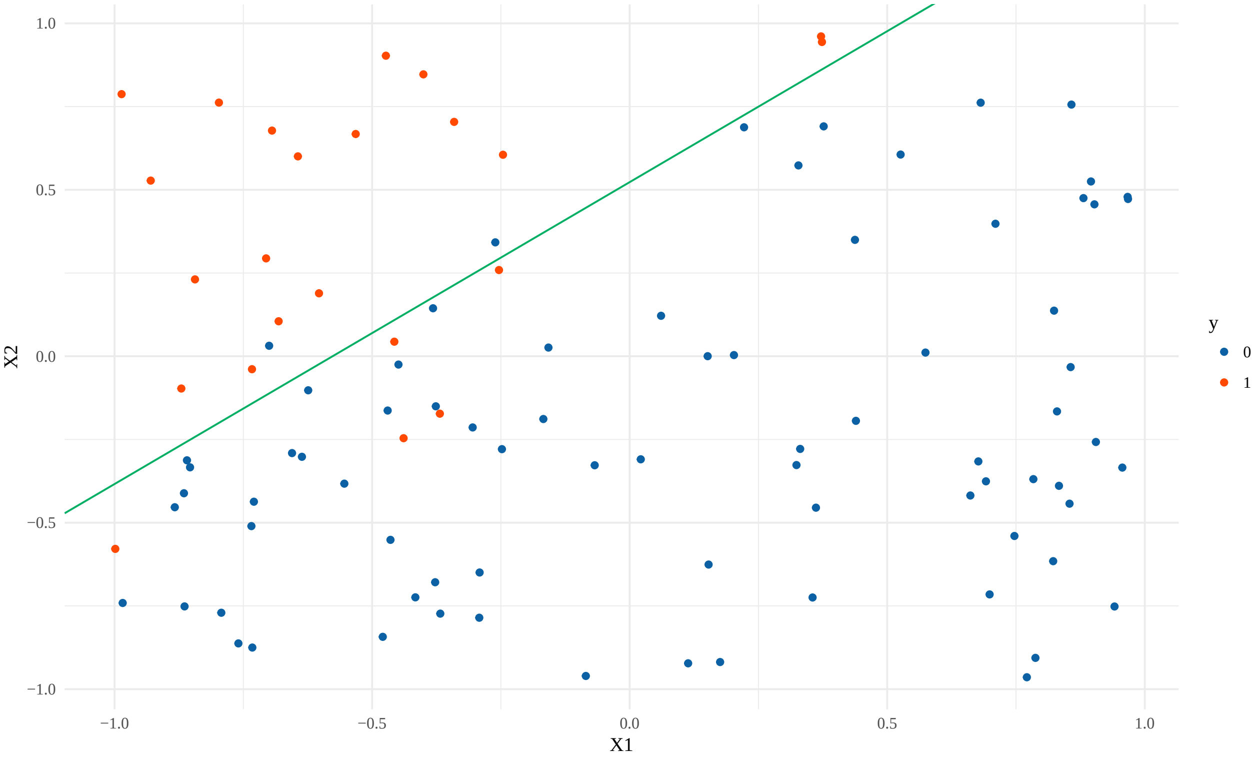
$$\frac{\exp\left(\widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_2\right)}{1+\exp\left(\widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_2\right)} > 0.5?$$

- Solving this gives

$$\widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_2 > \log(.5) - \log(1-.5) \Rightarrow x2 > -\frac{\widehat{\beta}_0 + \widehat{\beta}_1 x_1}{\widehat{\beta}_2}.$$

- That's just a line. Let's plot it:

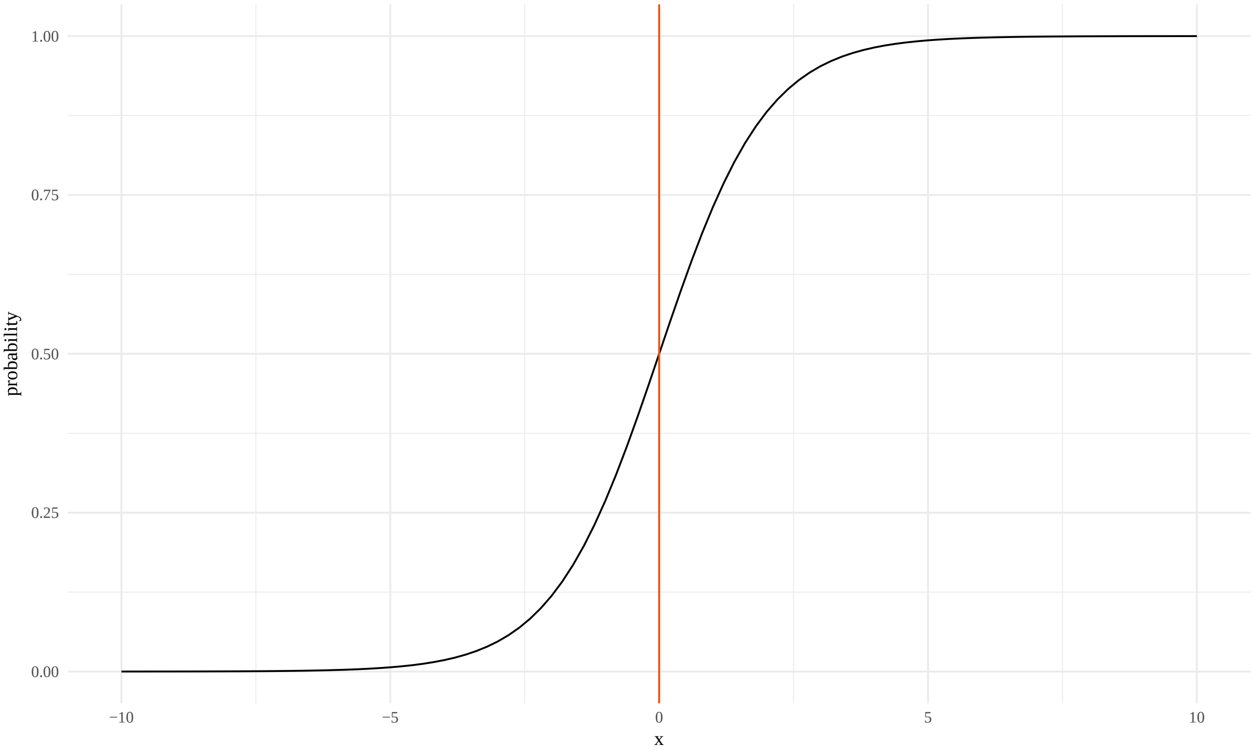
```r
cc = coefficients(log.lm)
g + geom_abline(intercept = -cc[1]/cc[3], slope = -cc[2]/cc[3], color=green)
```



Classification boundary

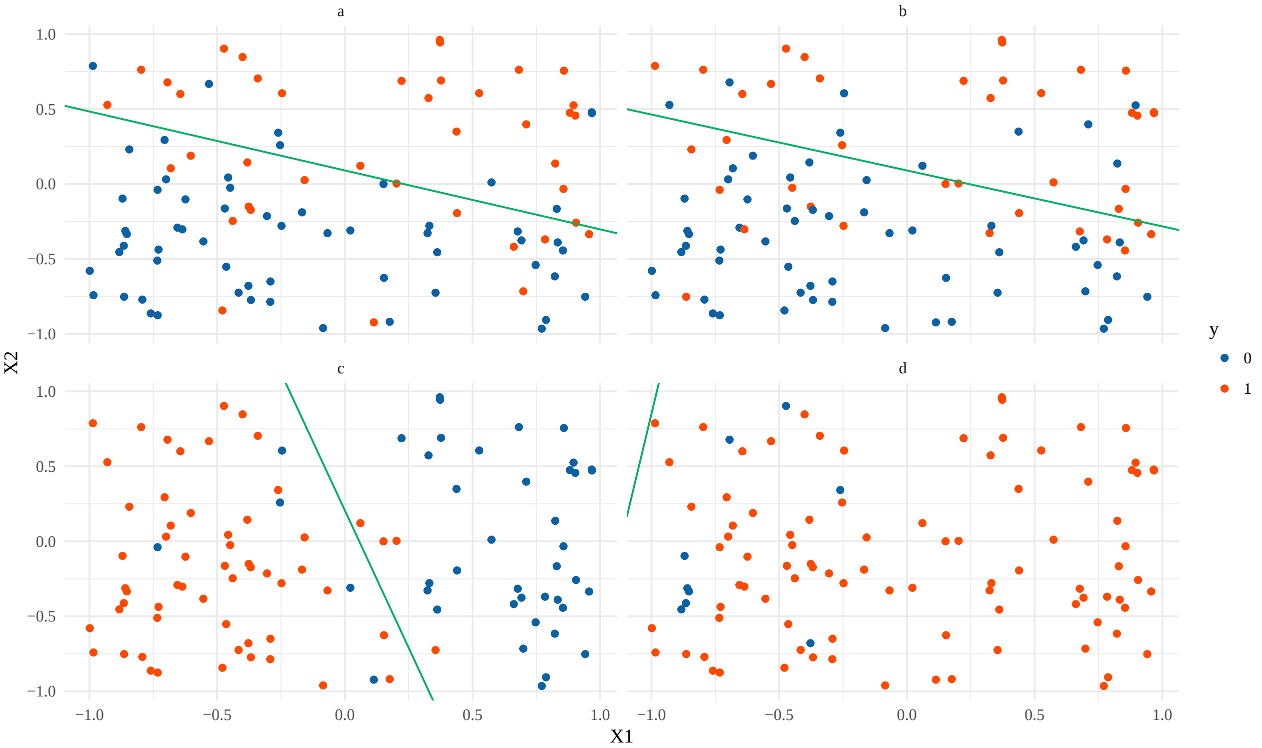
- We call that line the “classification boundary”
- The `ilogit` function looks like a ramp:

```
ggplot(data.frame(x=c(-10,10)), aes(x)) + stat_function(fun=ilogit) +
  geom_vline(xintercept = 0, color=red) + ylab('probability')
```



- Again, solving for where the equation crosses 0.5, gives us a “line”.
- Logistic regression always produces “linear” classification boundaries, so we call it a “linear classifier”

Lots of different boundaries



```
## # A tibble: 4 x 3
## # Groups: index [4]
##   index intercept  slope
##   <chr>     <dbl>  <dbl>
## 1 a        0.0907 -0.393
## 2 b        0.0905 -0.373
## 3 c        0.207  -3.67
## 4 d        7.96    7.11
```

Linear discriminant analysis

LDA

(Not to be confused with Latent Dirichlet Allocation, also abbrev. LDA)

- Suppose $X_i \mid Y_i = 1 \sim N(\mu_1, \Sigma)$
- And $X_i \mid Y_i = 0 \sim N(\mu_0, \Sigma)$
- Note that these share Σ.
- Suppose that $P(Y_i = 1) = \pi_1$

~~Bayes Rule:~~

$$P(Y_i = 1 \mid X_i = x) = \frac{P(X_i \mid Y_i = 1)\pi_1}{P(X_i \mid Y_i = 1)\pi_1 + P(X_i \mid Y_i = 0)\pi_0}$$

- So if we know $\pi_1, \mu_1, \mu_0, \Sigma$, then we can find $P(Y_i = 1 \mid X)$

Simplification

$$
\begin{aligned}
\frac{P(X_i \mid Y_i = 1)\pi_1}{P(X_i \mid Y_i = 1)\pi_1 + P(X_i \mid Y_i = 0)\pi_0} &= \frac{\pi_1 \frac{1}{(2\pi|\Sigma|)^{p/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^\top \Sigma^{-1}(x-\mu_1)\right)}{\sum_{j=0,1} \pi_j \frac{1}{(2\pi|\Sigma|)^{p/2}} \exp\left(-\frac{1}{2}(x-\mu_j)^\top \Sigma^{-1}(x-\mu_j)\right)} \\
&= \text{(take logs)} \\
&= \ldots \\
&= x^\top \Sigma^{-1}\mu_1 - \frac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1 + \log \pi_1 \\
&=: \delta_1
\end{aligned}
$$

- If this is bigger than δ_0, we predict ~~1~~ else ~~0~~.

Why is this linear?

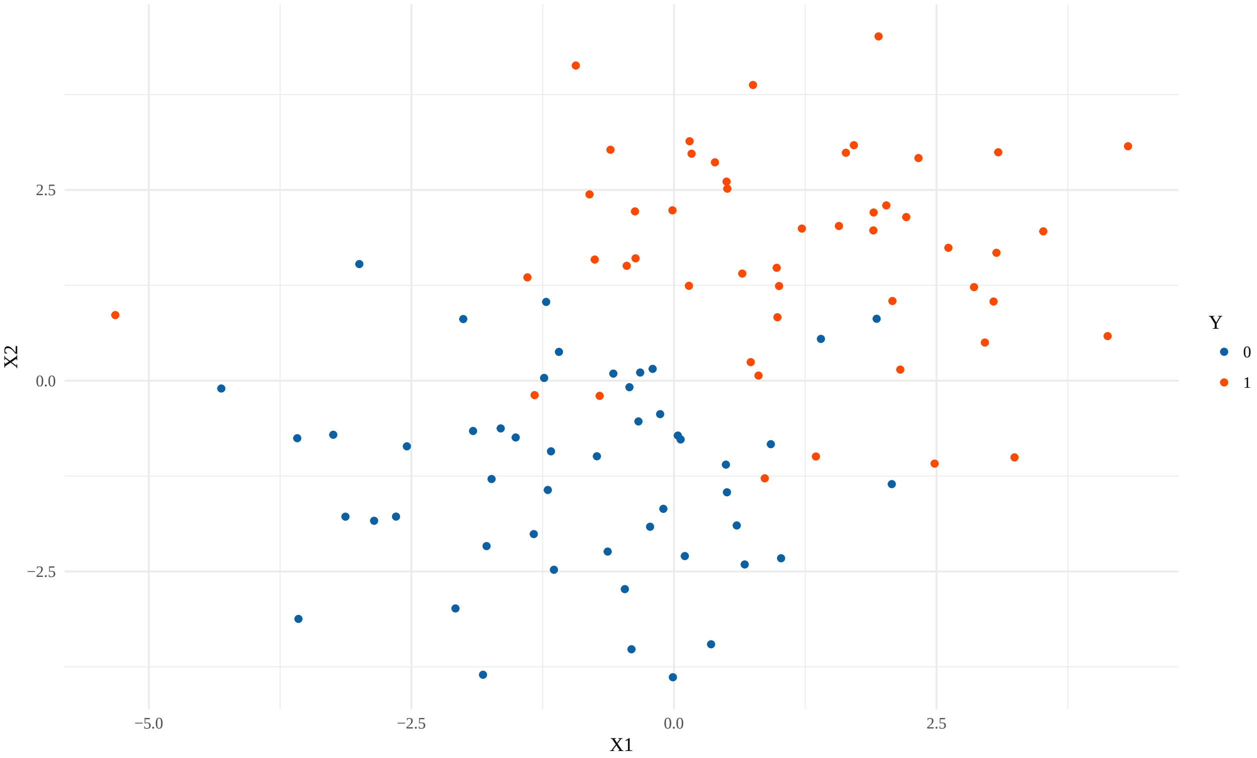
- We are indifferent when $\delta_1 = \delta_0$

$$
\begin{aligned}
&\delta_1 = \delta_0 \\
&\Rightarrow x^\top \Sigma^{-1}\mu_1 - \frac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1 + \log \pi_1 = x^\top \Sigma^{-1}\mu_0 - \frac{1}{2}\mu_0^\top \Sigma^{-1}\mu_0 + \log \pi_0 \\
&\Rightarrow x^\top \Sigma^{-1}(\mu_1 - \mu_0) - \frac{1}{2}\left(\mu_1^\top \Sigma^{-1}\mu_1 - \mu_0^\top \Sigma^{-1}\mu_0\right) + \log \pi_1 - \log \pi_0 = 0
\end{aligned}
$$

- The slope is $(\mu_1 - \mu_0)^\top \Sigma^{-1}$
- The intercept is $-\frac{1}{2}\left(\mu_0^\top \Sigma^{-1}\mu_0 - \mu_1^\top \Sigma^{-1}\mu_1\right) + \log \pi_1 - \log \pi_0$

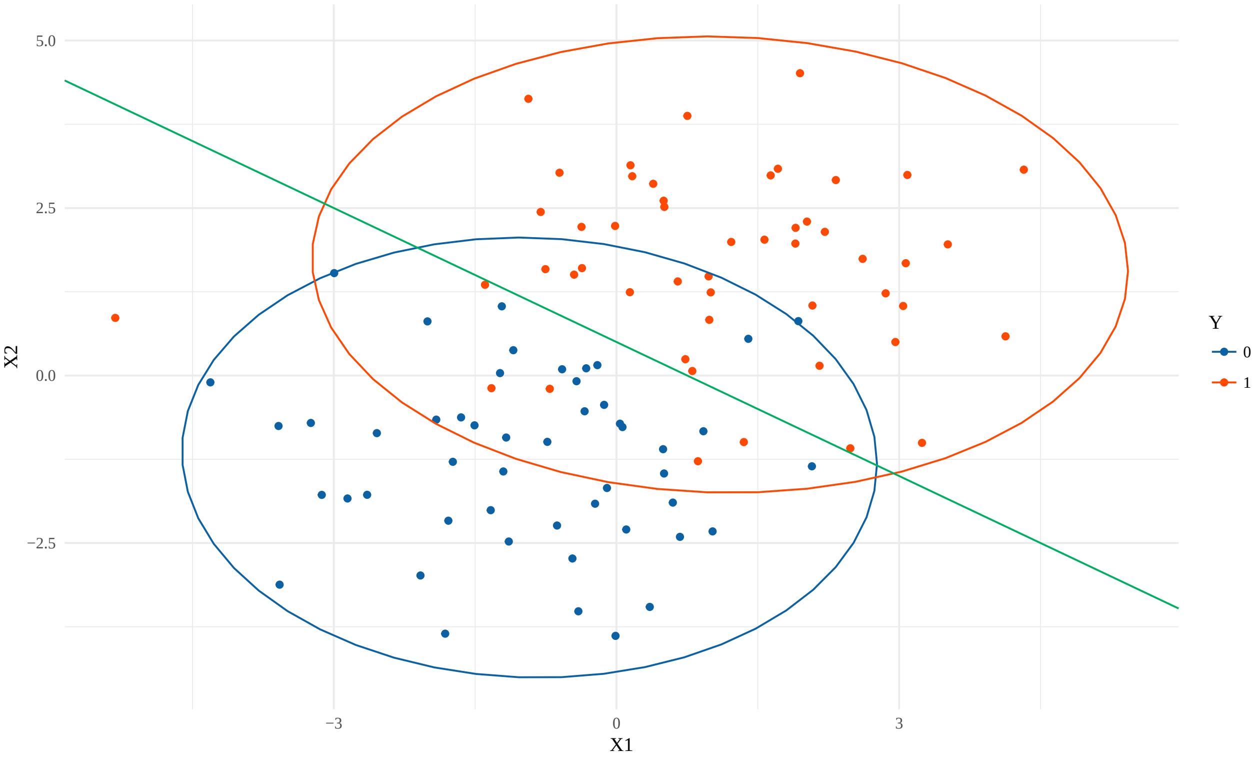
Example

```r
library(mvtnorm)
n = 100
pi1 = 0.5
n1 = floor(n*pi1); n0 = n-n1
mu1 = c(1,2); mu0 = c(-1,-1)
Sigma = 2*diag(2)
X1 = rmvnorm(n1, mu1, Sigma)
X2 = rmvnorm(n0, mu0, Sigma)
X = rbind(X1,X2)
Y = factor(c(rep(1,n1),rep(0,n0)))
df = data.frame(Y,X)
g <- ggplot(df, aes(X1,X2,color=Y)) + geom_point() + scale_color_manual(values=c(blue,red))
g
```



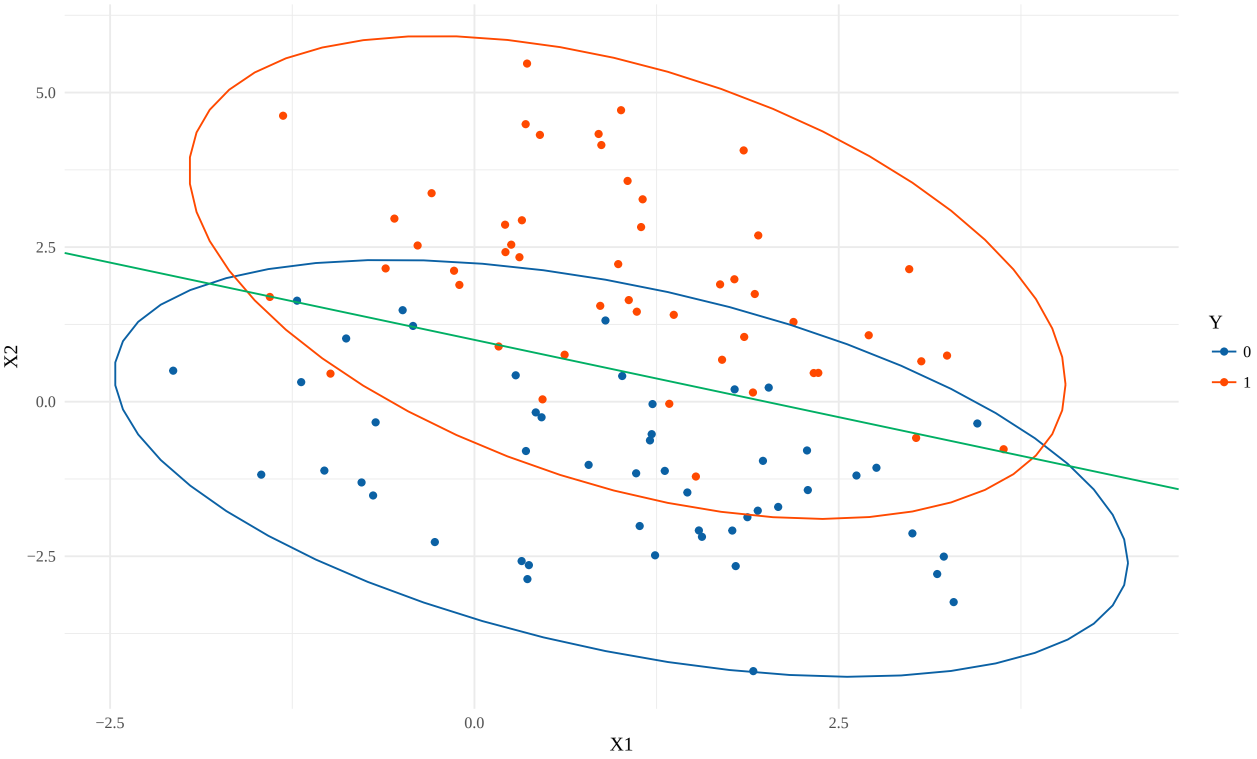
The distributions and the classifier

```
Sinv = solve(Sigma)
slope.vec = t(mu1-mu0) %*% Sinv
intercept = 0.5*(t(mu0) %*% Sinv %*% mu0 - t(mu1) %*% Sinv %*% mu1)
g + stat_ellipse(type='norm') + # these are estimated, not the truth
  geom_abline(intercept = -intercept/slope.vec[2],
              slope = -slope.vec[1]/slope.vec[2], color=green)
```



Try another one

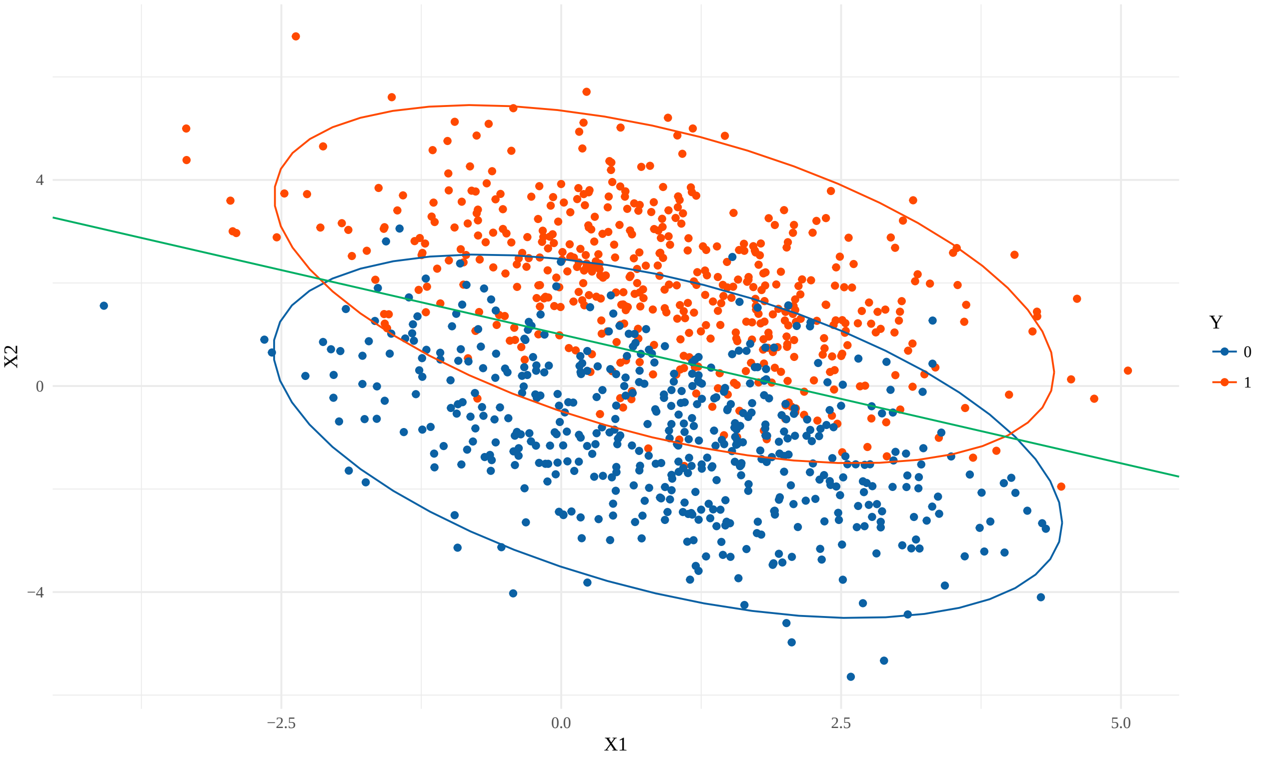
```
mu1 = c(1,2); mu0 = c(1,-1)
Sigma = 2*matrix(c(1,-.5,-.5,1),2)
X1 = rmvnorm(n1, mu1, Sigma)
X2 = rmvnorm(n0, mu0, Sigma)
X = rbind(X1,X2)
Y = factor(c(rep(1,n1),rep(0,n0)))
df = data.frame(Y,X)
Sinv = solve(Sigma)
slope.vec = t(mu1-mu0) %*% Sinv
intercept = 0.5*(t(mu0) %*% Sinv %*% mu0 - t(mu1) %*% Sinv %*% mu1)
ggplot(df, aes(X1,X2,color=Y)) + geom_point() + scale_color_manual(values=c(blue,red)) +
  stat_ellipse(type='norm') +
  geom_abline(intercept = -intercept/slope.vec[2],
              slope = -slope.vec[1]/slope.vec[2], color=green)
```



- Note: here there is a single Σ, but I don't know how to plot ellipses in `ggplot`. So these are estimated.

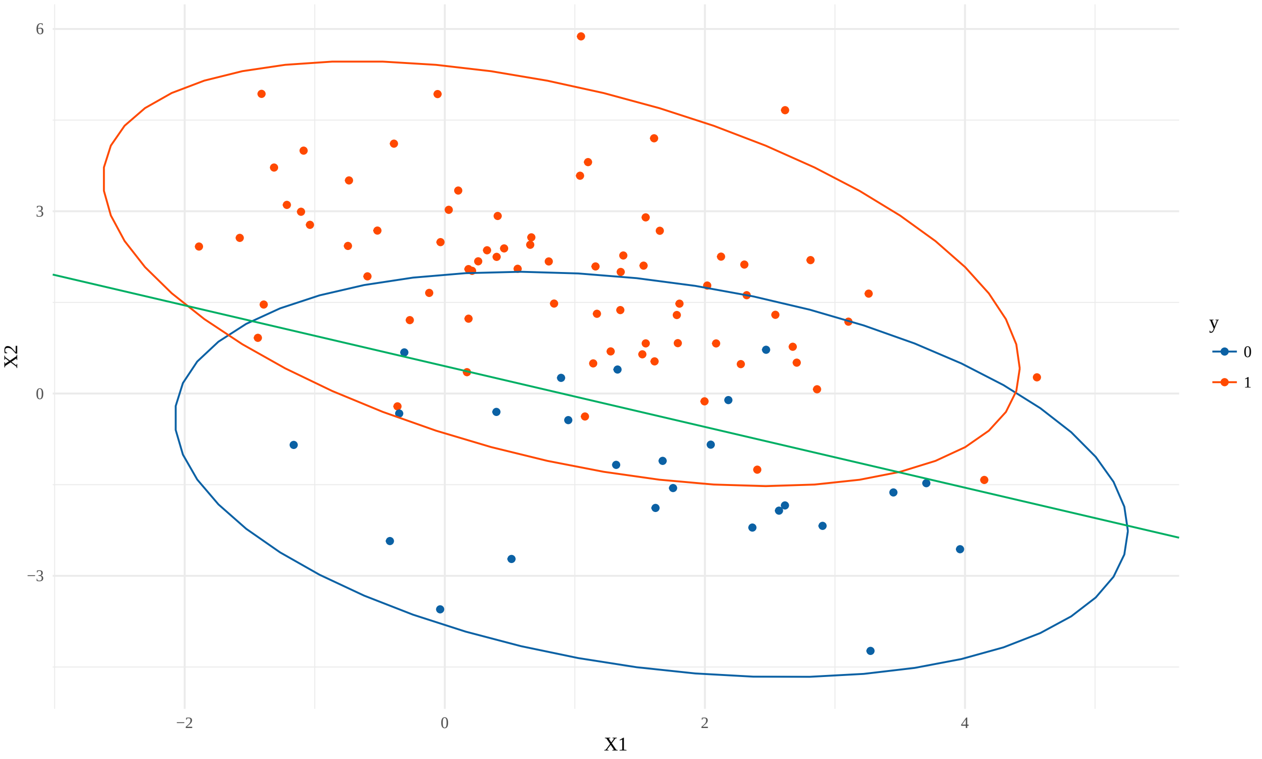
Same one, but make n big

```
n1=500; n0=500
X1 = rmvnorm(n1, mu1, Sigma)
X2 = rmvnorm(n0, mu0, Sigma)
X = rbind(X1,X2)
Y = factor(c(rep(1,n1),rep(0,n0)))
df = data.frame(Y,X)
Sinv = solve(Sigma)
slope.vec = t(mu1-mu0) %*% Sinv
intercept = 0.5*(t(mu0) %*% Sinv %*% mu0 - t(mu1) %*% Sinv %*% mu1)
ggplot(df, aes(X1,X2,color=Y)) + geom_point() + scale_color_manual(values=c(blue,red)) +
  stat_ellipse(type='norm') +
  geom_abline(intercept = -intercept/slope.vec[2],
              slope = -slope.vec[1]/slope.vec[2], color=green)
```



Same one, but change P(Y=1)

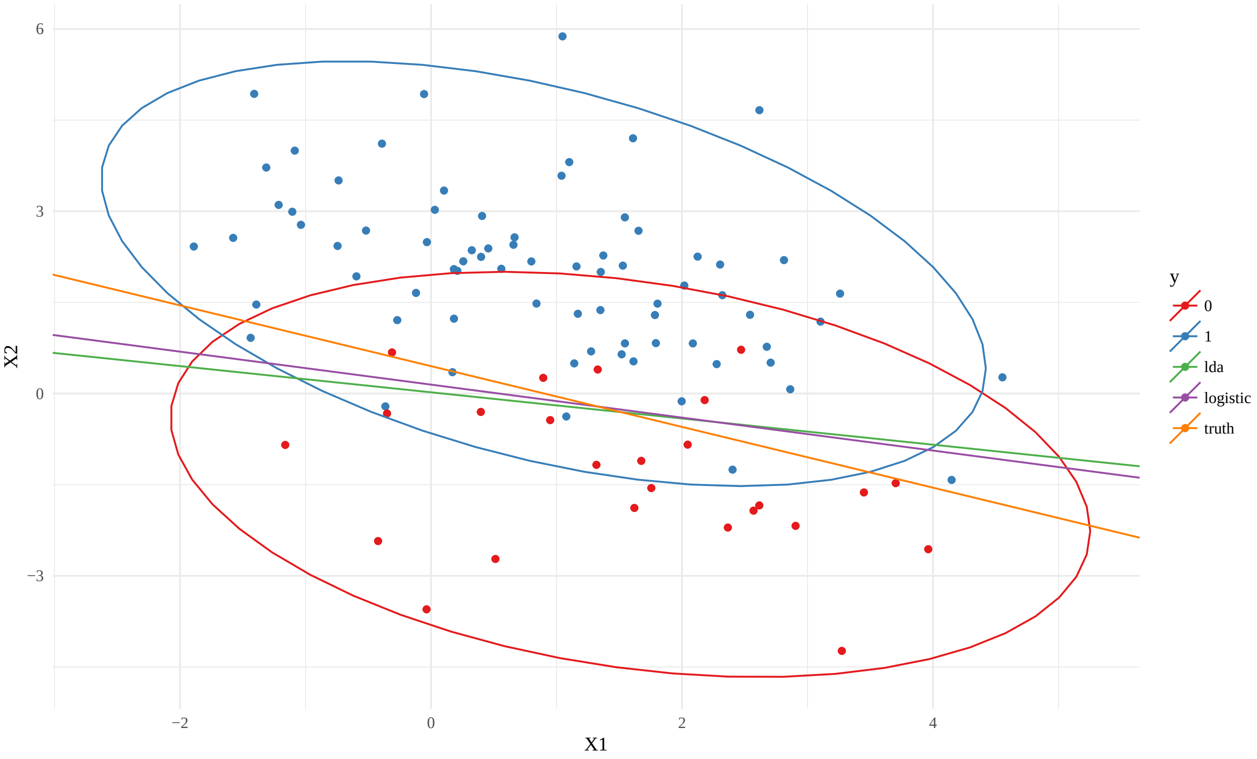
```
n1=75; n0=25
X1 = rmvnorm(n1, mu1, Sigma)
X2 = rmvnorm(n0, mu0, Sigma)
X = rbind(X1,X2)
y = factor(c(rep(1,n1),rep(0,n0)))
df = data.frame(y,X)
Sinv = solve(Sigma)
slope.vec = t(mu1-mu0) %*% Sinv
intercept = 0.5*(t(mu0) %*% Sinv %*% mu0 - t(mu1) %*% Sinv %*% mu1) + log(.75) - log(.25)
ggplot(df, aes(X1,X2,color=y)) + geom_point() + scale_color_manual(values=c(blue,red)) +
  stat_ellipse(type='norm') +
  geom_abline(intercept = -intercept/slope.vec[2],
              slope = -slope.vec[1]/slope.vec[2], color=green)
```



Ok, how do you do it?

- Estimate everything with sample analogues
- $\widehat{\pi}_1 = n_1/(n_1 + n_0)$
- $\widehat{\mu}_1 = \frac{1}{n_1}\sum X_i I(Y_i = 1)$. Same for $\widehat{\mu}_0$.
- $\widehat{\Sigma} = \frac{1}{n}\sum_{ij} I(Y_i = j)(X_i - \widehat{\mu}_j)^\top (X_i - \widehat{\mu}_j)$

```
library(MASS)
lda.fit = lda(y~X1+X2, data=df)
sl.int = lda.disc(lda.fit,df)
log.bd = decision.boundary(df)
truth = data.frame(intercept=-intercept/slope.vec[2], slope=-slope.vec[1]/slope.vec[2])
dfa = rbind(sl.int,log.bd,truth)
dfa$discriminant = c('lda','logistic','truth')
ggplot(df, aes(X1,X2,color=y)) + geom_point() + scale_color_brewer(palette = 'Set1')+
  stat_ellipse(type='norm') +
  geom_abline(mapping=aes(intercept=intercept, slope=slope,color=discriminant),data=dfa)
```



Comparing LDA and Logistic regression

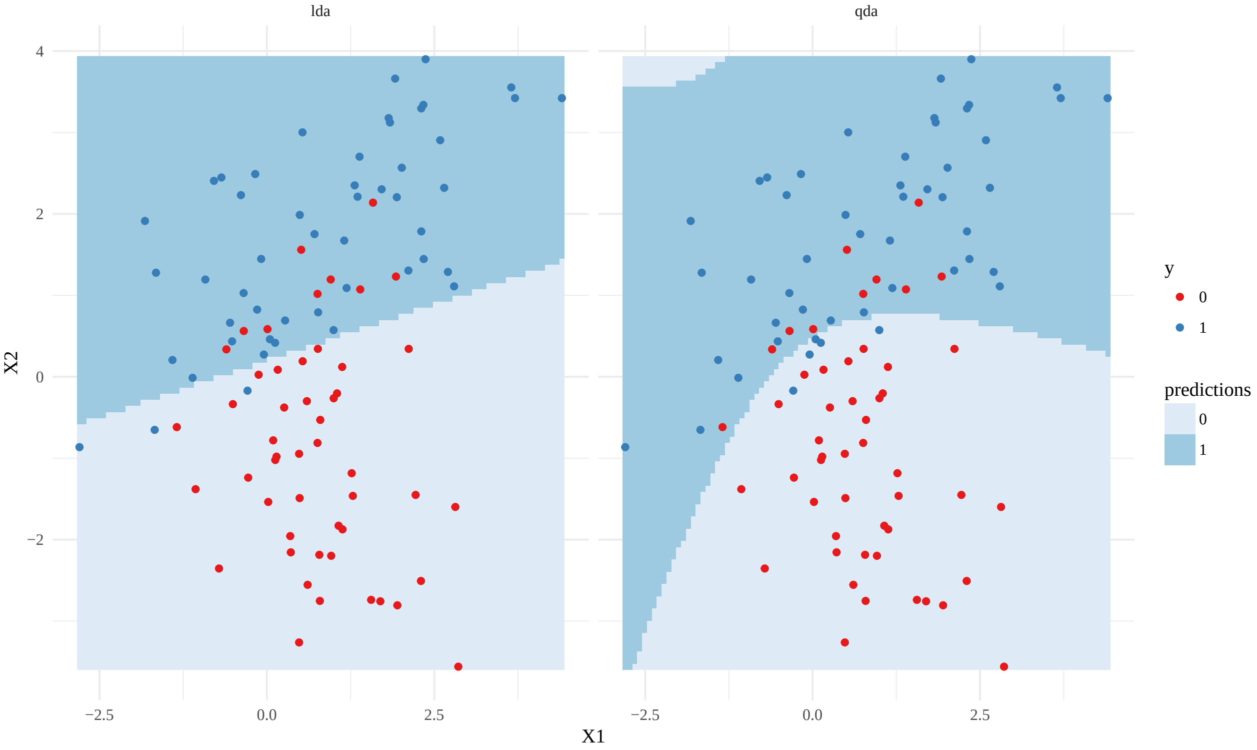
- Both are linear in x:
  - LDA$\longrightarrow \alpha_0 + \alpha_1^\top x$
  - Logit$\longrightarrow \beta_0 + \beta_1^\top x$.
- But the parameters are estimated differently.
- Examine the joint distribution of (X, y):
  - LDA $\prod_i f(x_i, y_i) = \underbrace{\prod_i f(X_i|y_i)}_{\text{Gaussian}} \underbrace{\prod_i f(y_i)}_{\text{Bernoulli}}$
  - Logistic $\prod_i f(x_i, y_i) = \underbrace{\prod_i f(y_i|X_i)}_{\text{Logistic}} \underbrace{\prod_i f(X_i)}_{\text{Ignored}}$
- LDA estimates the joint, but Logistic estimates only the conditional distribution. But this is really all we need.
- So logistic requires fewer assumptions.
- But if the two classes are perfectly separable, logistic crashes (and the MLE is undefined)
- LDA works even if the conditional isn't normal, but works poorly if any X is qualitative

QDA

- Start like LDA, but let $\Sigma_1 \neq \Sigma_0$.
- This gives a "quadratic" decision boundary (it's a curve).

- If we have many columns in X (p)
    - Logistic estimates $p+1$ parameters
    - LDA estimates $2p + p(p+1)/2 + 1$
    - QDA estimates $2p + p(p+1) + 1$
- If $p = 50$,
    - Logistic: 51
    - LDA: 1376
    - QDA: 2651
- QDA doesn't get used much: there are better nonlinear versions with way "fewer" parameters (SVMs)
- LDA only really depends on $\Sigma^{-1}(\mu_1 - \mu_0)$ and $(\mu_1 + \mu_0)$, so it has $< 2p$ parameters.

```
n1=50; n0=50
Sigma1 = matrix(c(2,.8,.8,1),2)
Sigma0 = matrix(c(1,-.5,-.5,2),2)
X1 = rmvnorm(n1, mu1, Sigma1)
X2 = rmvnorm(n0, mu0, Sigma0)
X = rbind(X1,X2)
y = factor(c(rep(1,n1),rep(0,n0)))
df = data.frame(y,X)
qda.fit = qda(y~X1+X2, data=df)
lda.fit = lda(y~X1+X2, data=df)
pred.grid = expand.grid(X1=seq(min(df$X1),max(df$X1),len=100),
                        X2=seq(min(df$X2),max(df$X2),len=100))
pred.grid$qda = predict(qda.fit, newdata=pred.grid)$class
pred.grid$lda = predict(lda.fit, newdata=pred.grid)$class
pg = gather(pred.grid,key='key',value='predictions',-c(X1,X2))
ggplot(pg, aes(X1,X2)) + geom_raster(aes(fill=predictions)) +
  facet_wrap(~key) + scale_fill_brewer()+
  geom_point(data=df,mapping=aes(X1,X2,color=y)) +
  scale_color_brewer(palette = 'Set1')
```

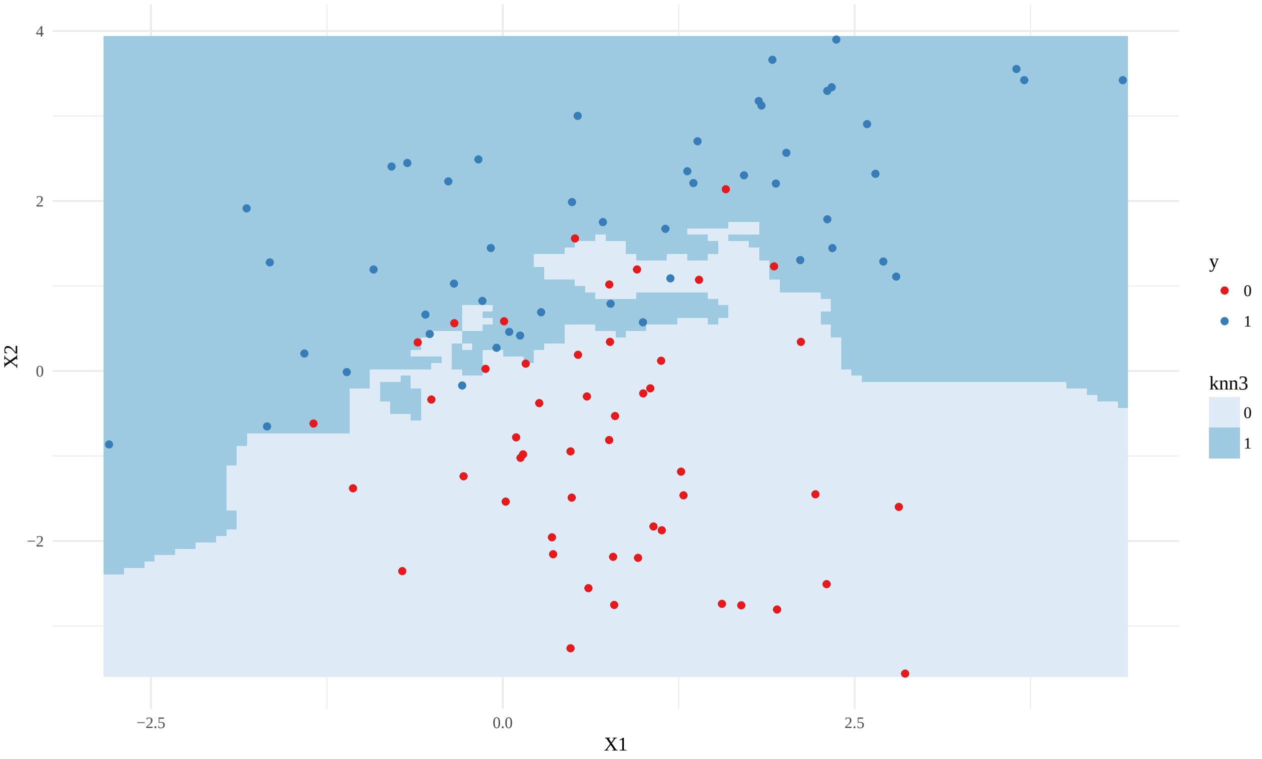


KNN

Re-entry

- We saw *k*-nearest neighbors at the very beginning of the course.

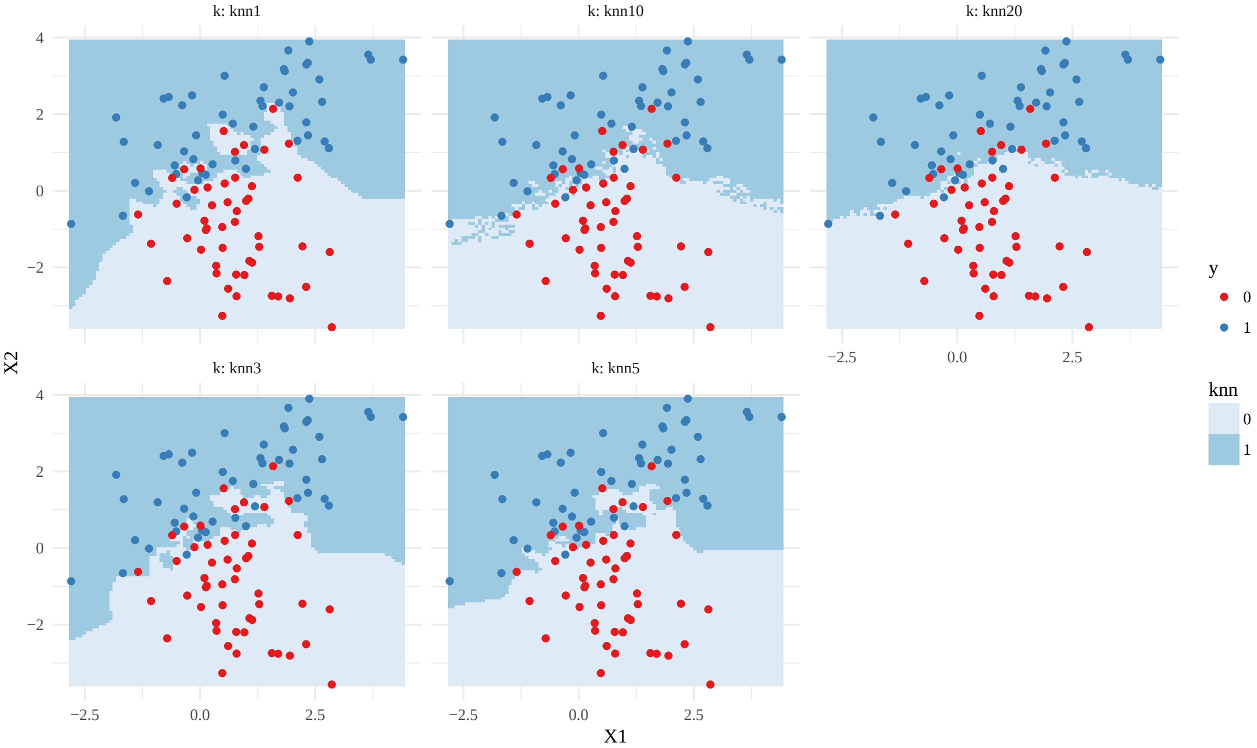
```
library(class)
pred.grid = expand.grid(X1=seq(min(df$X1),max(df$X1),len=100),
                        X2=seq(min(df$X2),max(df$X2),len=100))
pred.grid$knn3 = knn(df[,-1], pred.grid, df$y, k=3)
ggplot(pred.grid, aes(X1,X2)) + geom_raster(aes(fill=knn3)) +
  scale_fill_brewer() + geom_point(data=df,mapping=aes(X1,X2,color=y)) +
                        scale_color_brewer(palette = 'Set1')
```



Choosing k

- Choosing k is very important.

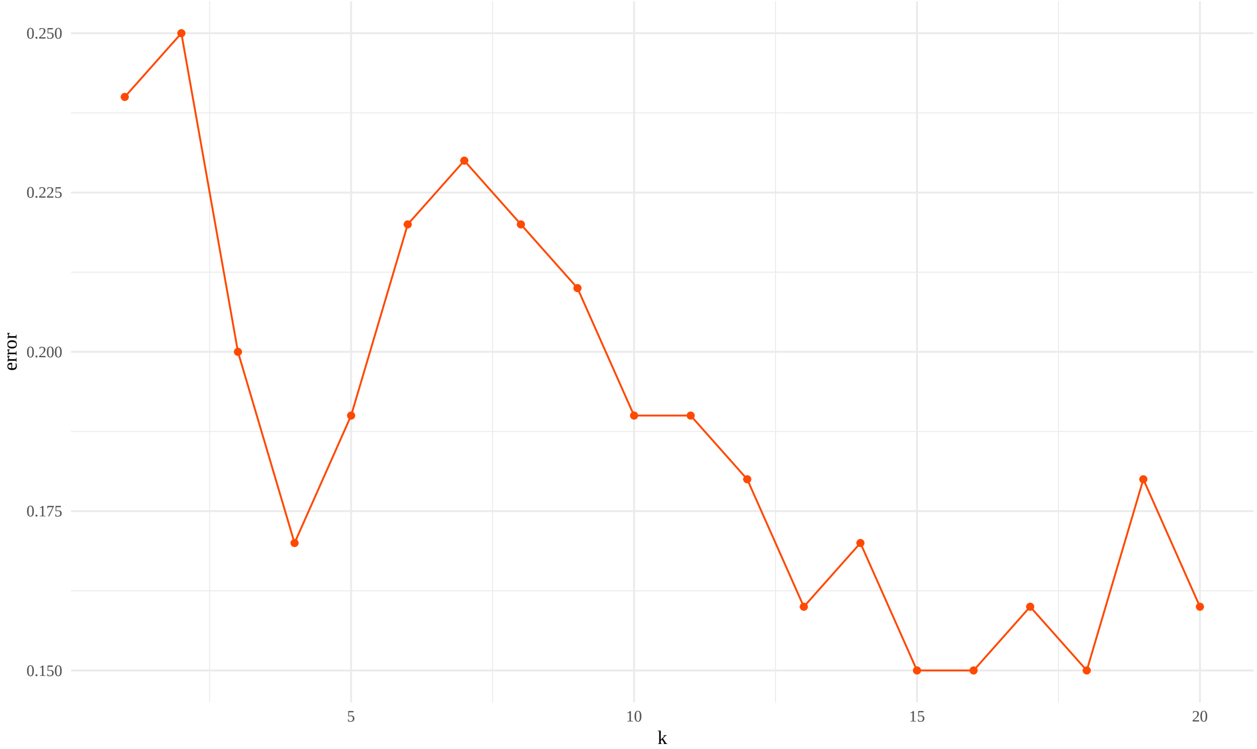
```
pred.grid$knn1 = knn(df[,-1], pred.grid[,1:2], df$y, k=1)
pred.grid$knn5 = knn(df[,-1], pred.grid[,1:2], df$y, k=5)
pred.grid$knn10 = knn(df[,-1], pred.grid[,1:2], df$y, k=10)
pred.grid$knn20 = knn(df[,-1], pred.grid[,1:2], df$y, k=20)
pg = gather(pred.grid,key='k',value='knn',-c(X1,X2))

ggplot(pg, aes(X1,X2)) + geom_raster(aes(fill=knn)) +
  facet_wrap(~k,labeller = label_both) + scale_fill_brewer()+
  geom_point(data=df,mapping=aes(X1,X2,color=y)) +
  scale_color_brewer(palette = 'Set1')
```



- How should we choose k?
- Scaling is also very important. The nearest neighbors are determined by their distance, so better to standardize your data first.

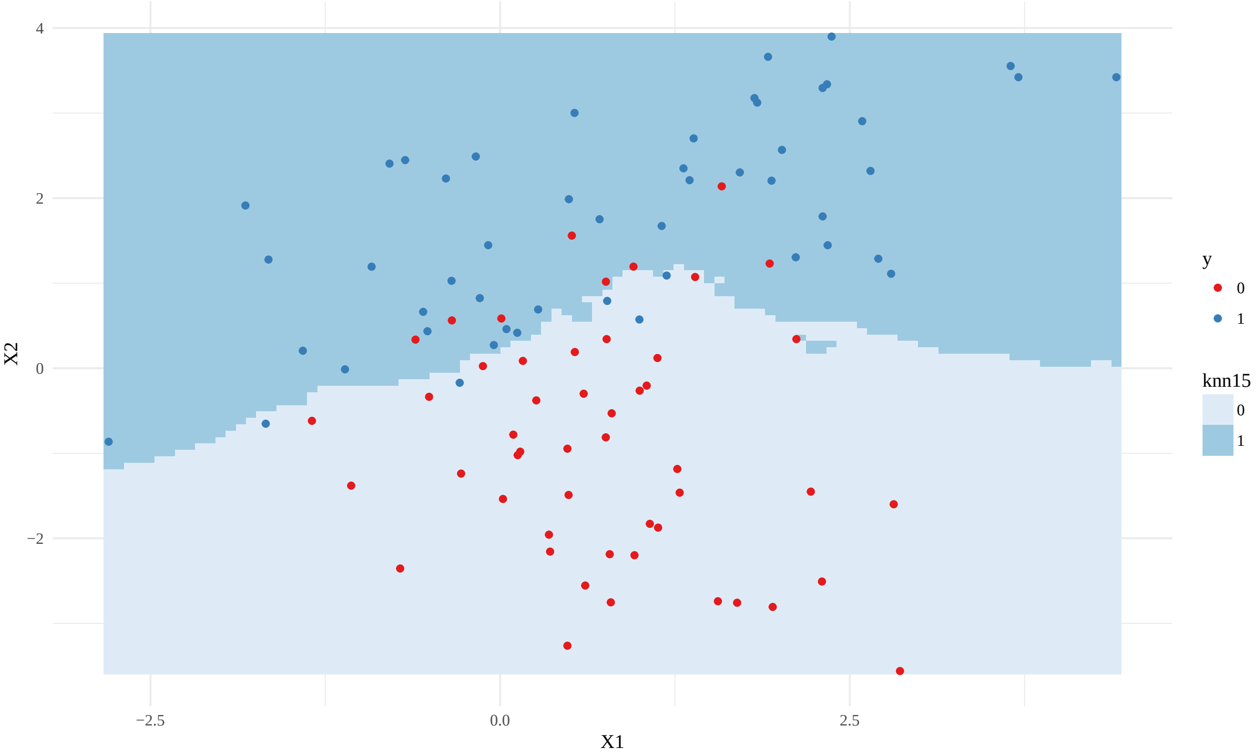
knn.cv

```
kmax = 20
err = double(kmax)
for(ii in 1:kmax){
  pk = knn.cv(df[,-1],df$y, k=ii) # does leave one out CV
  err[ii] = mean(pk != df$y)
}
ggplot(data.frame(k=1:kmax,error=err), aes(k,error)) + geom_point(color=red) +
  geom_line(color=red)
```



- I would use the ~~largest~~ `k` that is close to the minimum. This produces simpler, smoother, decision boundaries.

```
pred.grid$knn15 = knn(df[,-1], pred.grid[,1:2], df$y, k=15)
ggplot(pred.grid, aes(X1,X2)) + geom_raster(aes(fill=knn15)) +
  scale_fill_brewer() + geom_point(data=df,mapping=aes(X1,X2,color=y)) +
                        scale_color_brewer(palette = 'Set1')
```



```
(tt <- table(knn(df[,-1],df[,-1],df$y,k=15),df$y,dnn=c('predicted','truth')))
```

```
##          truth
## predicted  0  1
##         0 42  5
##         1  8 45
```

```
1-sum(diag(tt))/sum(tt)
```

```
## [1] 0.13
```

Kernelization

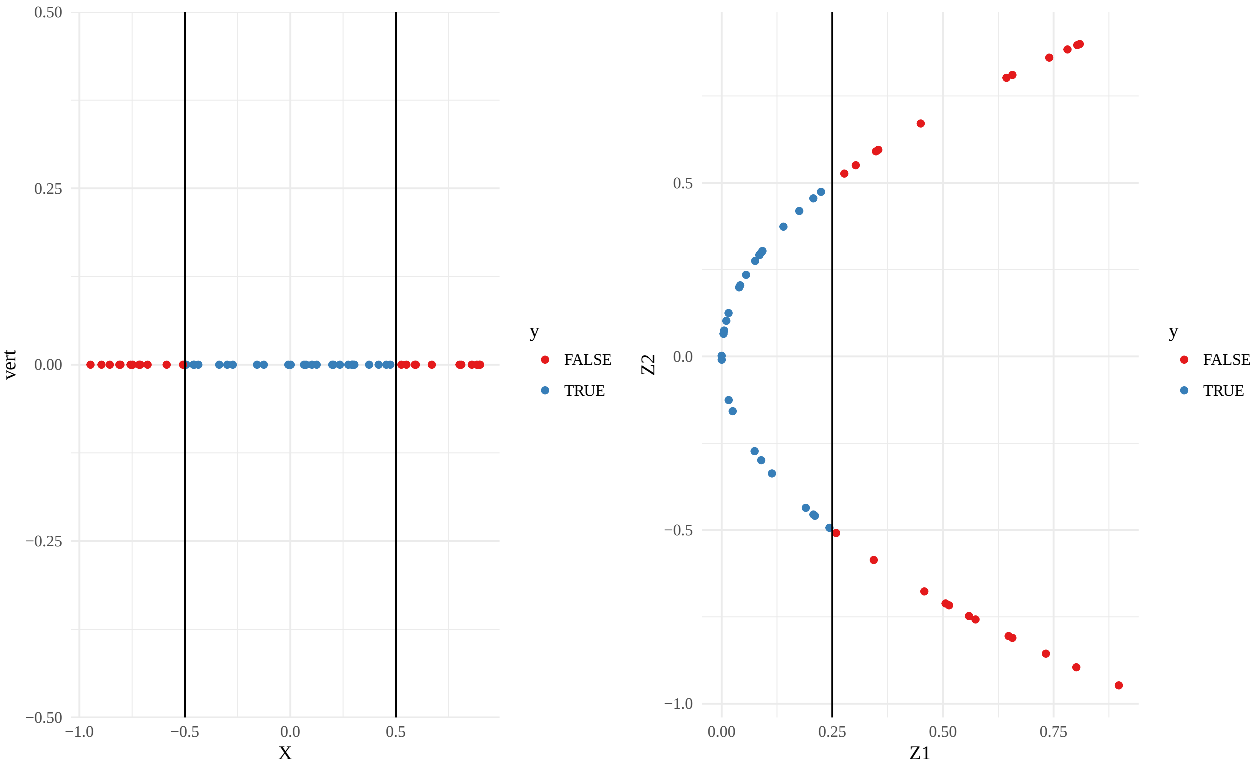
Other non-linear classifiers

- We already saw that knn produces non-linear decision boundaries.
- Next week, we'll see trees, which are also non-linear.
- "Kernelization" is a way of turning linear classifiers (or linear regression) into non-linear classifiers.
- You've already seen this happen when you add interactions or quadratic terms to linear models

The idea

- Suppose X_i takes values in some space $\mathcal{X}$.
- Find a mapping $\phi : \mathcal{X} \to \mathcal{Z}$.
- Apply a linear classifier on $\mathcal{Z}$.

- Example: $\mathcal{X} = \mathbb{R}$, $\mathcal{Z} = \mathbb{R}^2$, $\phi(x) = (z_1, z_2) = (x, x^2)$



- A linear classifier in the higher-dimensional space corresponds to a non-linear classifier in low dimensions.
- Of course, if $\dim(\mathcal{Z})$ is too big, then there will be too many parameters to estimate well.

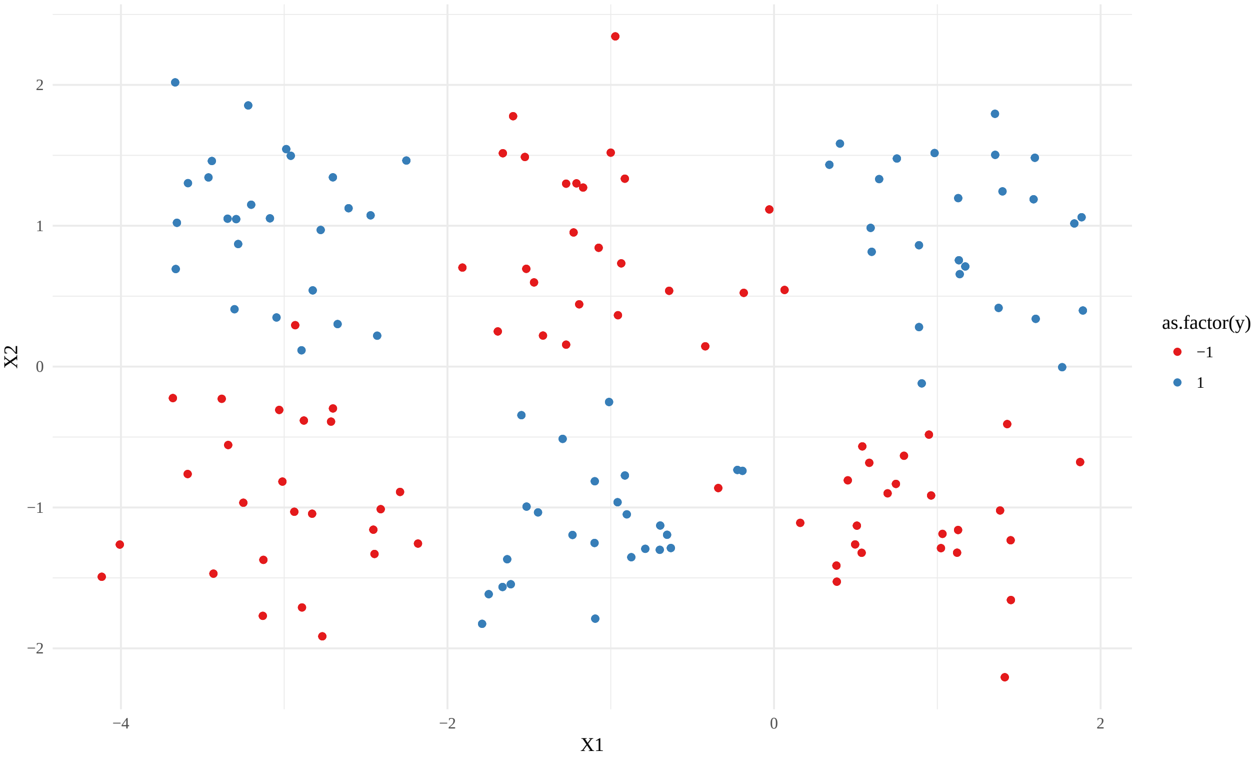
The trick

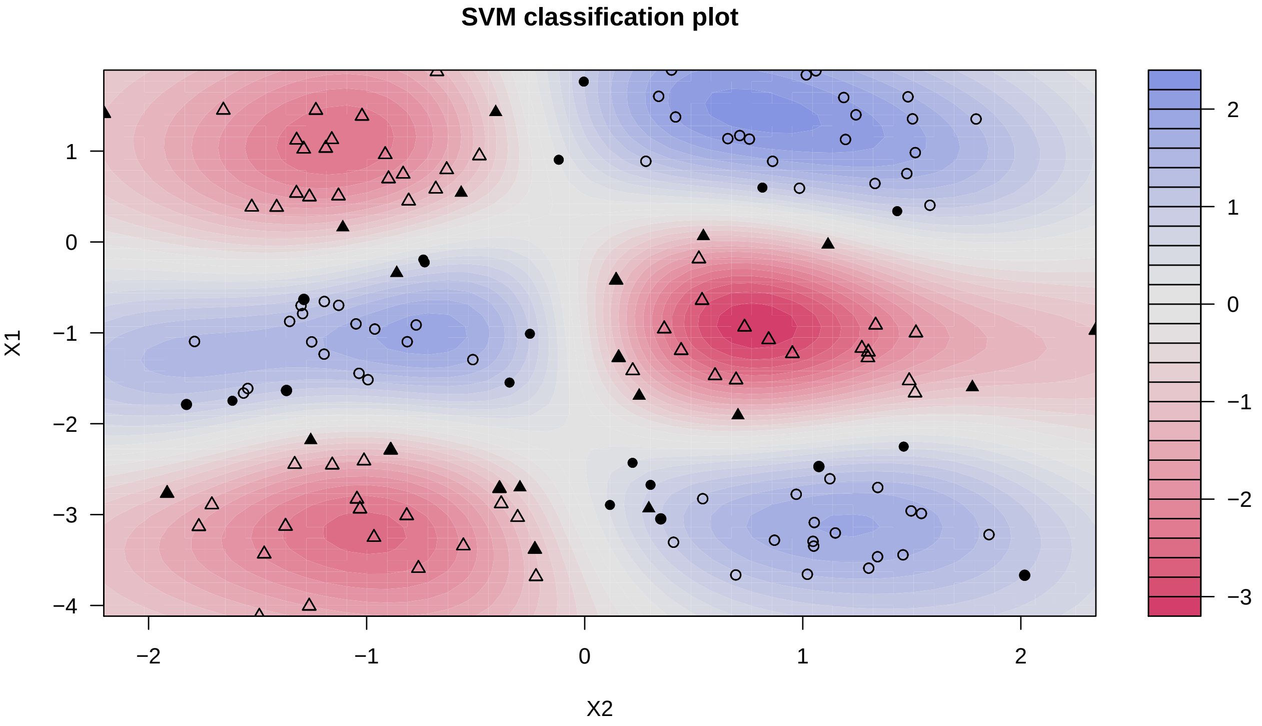
If:

1. There is a mapping $\phi : \mathcal{X} \to \mathcal{Z}$
2. Your classifiers of choice only needs the ~~inner product~~ between observations, not the observations themselves. (logistic regression, LDA, and SVMs work)
3. There is a function K such that $K(x, x') = \langle \phi(x), \phi(x') \rangle$ (Mercer's theorem gives this)

Then, we can just replace all inner products $\langle x, x' \rangle$ with $K(x, x')$.

- This produces a nonlinear classifier based on a linear classifier.
- We don't actually need x or ϕ, just $K(x, x')$ (this is an $n \times n$ matrix)
- $\mathcal{Z}$ can be infinite dimensional.


as.factor(y)
−1
1
X2
2
1
0
−1
−2
X1
−4
−2
0
2


SVM classification plot
X1
1
0
−1
−2
−3
−4
X2
−2
−1
0
1
2
2
1
0
−1
−2
−3