Lab03: SVM and Kernel Intuition

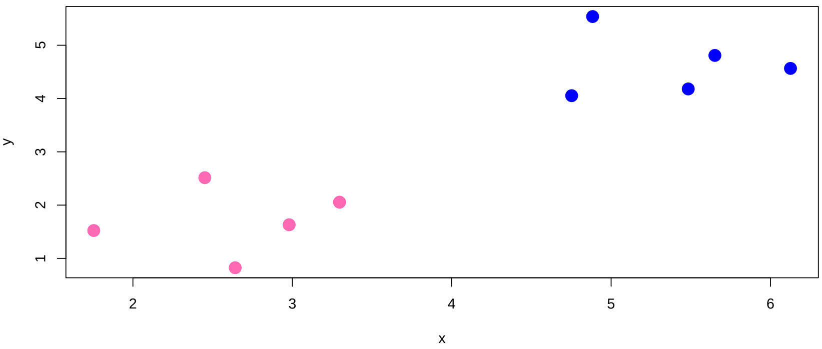
Today

1. Recall SVM when linearly separable
2. SVM when not linearly separable (we'll want to use a kernel)
3. SVM in R

Recall

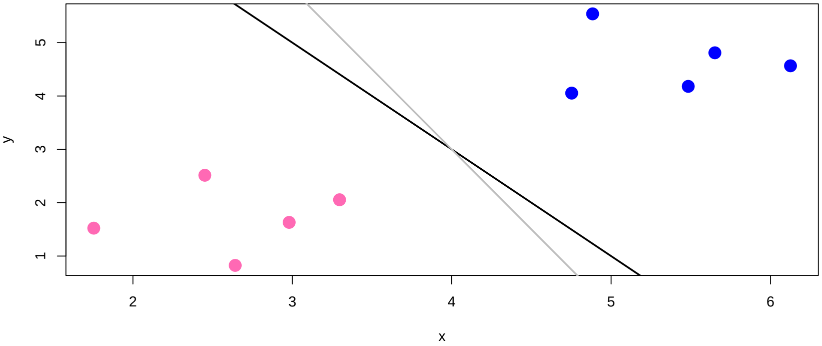
Recall SVM find the hyperplane (a line in two dimensions) that best separates the data

```r
set.seed(0406)
x <- c(rnorm(5, 2, .5), rnorm(5, 5, .5))
y <- c(rnorm(5, 2, .5), rnorm(5, 5, .5))
plot(x = x, y = y, col = c(rep("hotpink", 5), rep("blue", 5)), pch = 16
```



Which line?

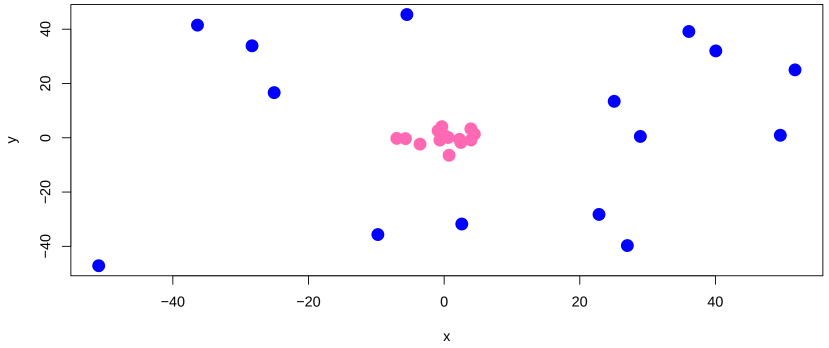
```
plot(x = x, y = y, col = c(rep("hotpink", 5), rep("blue", 5
abline(11,-2, lwd = 2)
abline(15,-3, col = "grey", lwd = 2)
```



Of course, data aren't usually this friendly

- Our classes aren't usually linearly separable
- But in this case, its obvious they ought to be easily separable

```r
set.seed(0406)
ang <- 2*pi*rnorm(30) #random angle
r <- c(2*rnorm(15, 2, 1), 10*rnorm(15, 5, 1)) #random radius (different
x <- r * cos(ang) ## x coord
y <- r * sin(ang) ## y coord
plot(x = x, y = y, col = c(rep("hotpink", 15), rep("blue", 15)), pch =
```

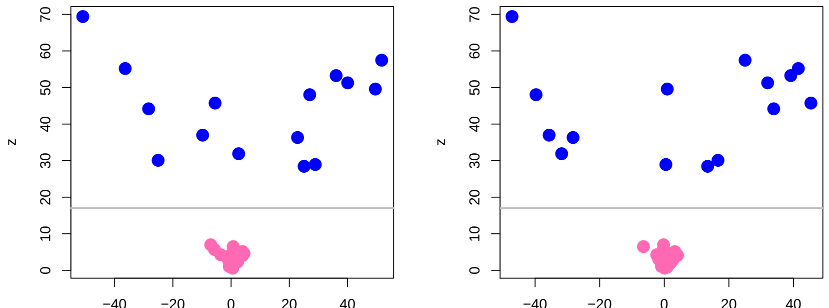


Add another dimension

Let's add $z = \sqrt{x^2 + y^2}$

- The hyperplane is now parallel to x/y axis at a certain value of z.

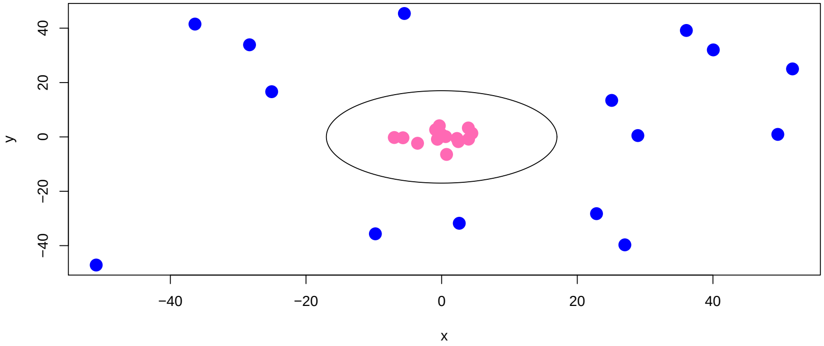
```r
z <- sqrt(x^2 + y^2)
par(mfrow = c(1,2))
plot(x = x, y = z, col = c(rep("hotpink", 15), rep("blue", 15)), pch =
abline(h = 17, col = "grey", lwd = 2)
plot(x = y, y = z, col = c(rep("hotpink", 15), rep("blue", 15)), pch =
abline(h = 17, col = "grey", lwd = 2)
```



Now what?

We need to map our separating hyperplane back into the dimensions of the data

```
plot(x = x, y = y, col = c(rep("hotpink", 15), rep("blue",
radius <- 17
theta <- seq(0, 2*pi, length = 200)
lines(x = radius * cos(theta), y = radius * sin(theta))
```



The key point of kernels for SVM

- We've seen we can classify nonlinear data by mapping our space into a higher dimension
- *But*, this can be computationally expensive for every vector in the data
- SVM doesn't need the *actual vectors* transformed to the higher dimensional space to find the optimal hyperplane, it only needs their *dot product*, bypassing expensive calculations of the new dimensions

Dot product intuition

Recall from lecture, the kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ (function that gives the inner product in the higher dimensional space) is a part of our optimization problem:

$$L(\lambda_i) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j}^{N} \lambda_i \lambda_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

Recall λ_i's are positive, and $L(\lambda)$ will be maximized if we give nonzero values to λ_i's that correspond to the support vectors.

So how does the kernal/inner product matter?

Dot product intuition

$$L(\lambda_i) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j}^{N} \lambda_i \lambda_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

Consider cosine similarity $k(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^T \mathbf{x}_j}{||\mathbf{x}_i|| \, ||\mathbf{x}_j||}$

1. If $\mathbf{x}_i, \mathbf{x}_j$ are **orthogonal/perpendicular**, their cosine similarity is 0 $\rightarrow$ should not contribute to L
2. If $\mathbf{x}_i, \mathbf{x}_j$ are **identical**, their cosine similarity is 1:
   - Could happen if both $\mathbf{x}_i, \mathbf{x}_j$ predict the same class $(y_i = y_j = 1)$ $\rightarrow$ detract from L $\rightarrow$ algorithm downgrades similar vectors that make the same prediction
   - Could happen if both $\mathbf{x}_i, \mathbf{x}_j$ predict the different classes (e.g., $y_i = 1 y_j = -1$) $\rightarrow$ add to L (subtracting negative second term, thus adding to the total sum) $\rightarrow$ algorithm searches for similar vectors that make opposite predictions

In R

Predict whether a cancer is malignant or benign from biopsy details

```r
library(mlbench)
data(BreastCancer)
table(BreastCancer$Class)
```

```
##
##    benign malignant
##       458       241
```

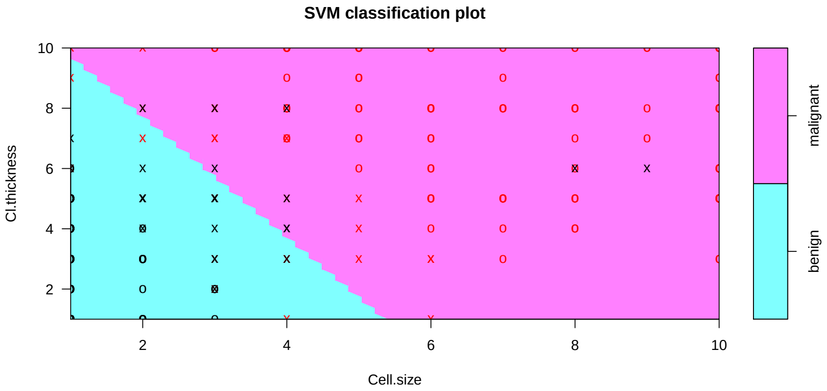
In R

Let's first try linear separation, then other kernels.

```r
library(e1071)
set.seed(0406)
BreastCancer <- BreastCancer[complete.cases(BreastCancer), ]
train_idx <- sample(1:nrow(BreastCancer), 400)
test_idx <- (1:nrow(BreastCancer))[-train_idx]
d <- BreastCancer[train_idx, c(2,3,11)]
d[,1] <- as.numeric(d[,1])
d[,2] <- as.numeric(d[,2])
mod <- svm(Class ~ ., data = d, kernel = 'linear')
#str(mod)
```

Plot results

```
#plot(x = d[,1], y = d[,2], col = ifelse(y == "benign", "blue", "hotpin
#points(t(t(mod$SV) * mod$x.scale[['scaled:scale']] + mod$x.scale[['sca
plot(mod, d)
```



Note

Drawn from James Le's tutorial on datacamp

- https://www.datacamp.com/community/tutorials/support-vector-machines-r

and Robert Berwick's slides

- http://web.mit.edu/6.034/wwwbob/svm.pdf