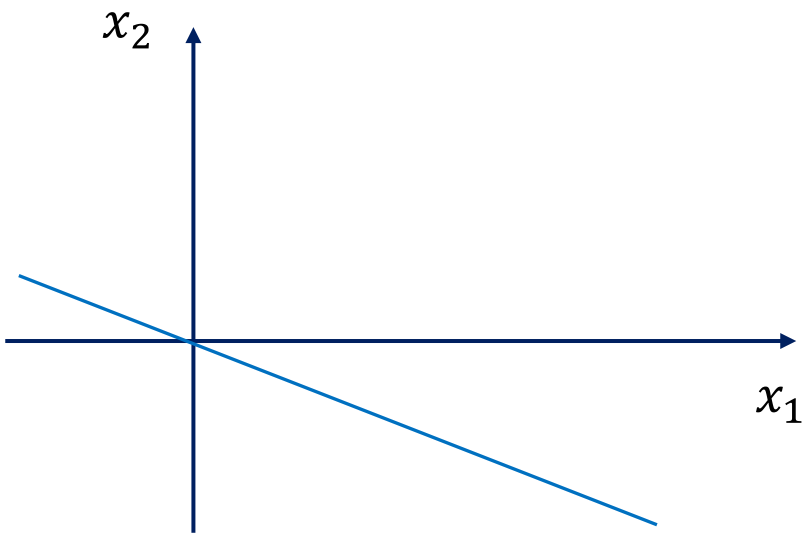
Kernel SVM

Hyperplane

- In a p-dim feature space, a hyperplane has the form

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0$$



E.g. 2-dim space

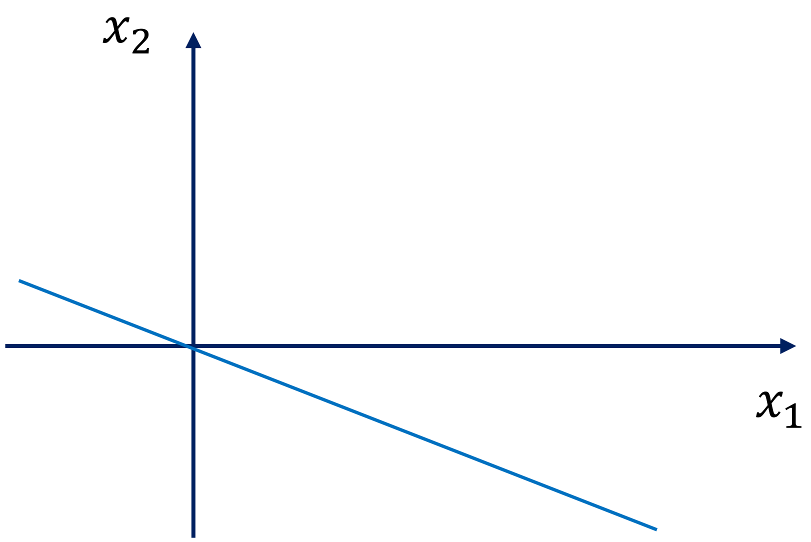
$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

What is $(\beta_0, \beta_1, \beta_2)$?

Hyperplane

- Distance to hyperplane (when $\beta_0 = 0$)   E.g. $(x_1, x_2) = (2,0)$

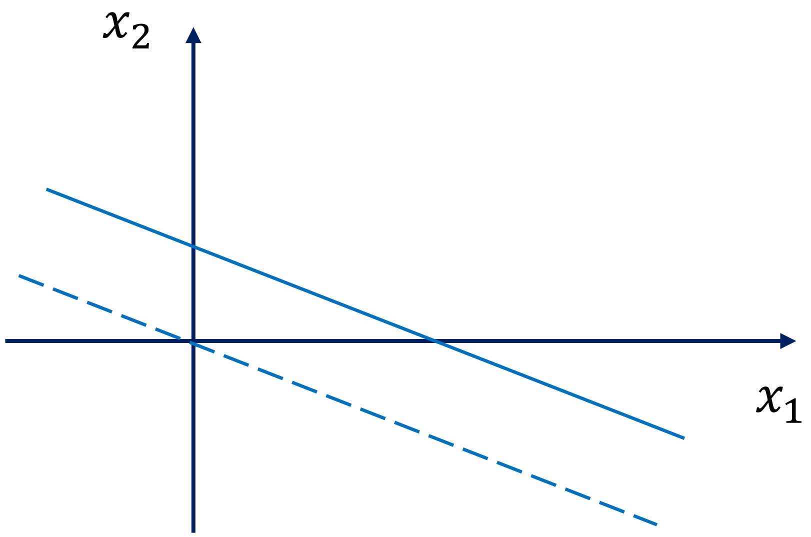
$$\vec{x} \cdot \vec{\beta}$$



- Note: Distance can be negative

Hyperplane

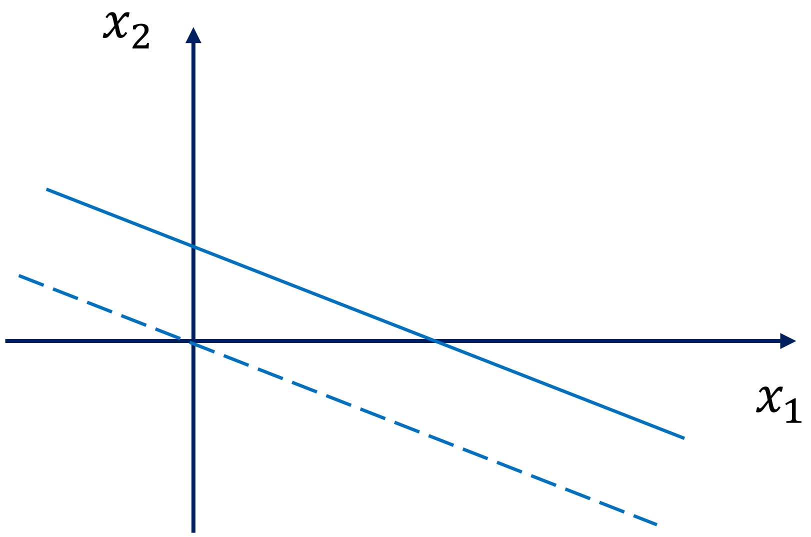
- When $\beta_0 \neq 0$: shift the hyperplane along the direction of normal vector by $|\beta_0|$



Hyperplane

- Distance to hyperplane (when $\beta_0 \neq 0$)

$$\vec{x} \cdot \vec{\beta} + \beta_0$$



- Note: Again, distance can be negative

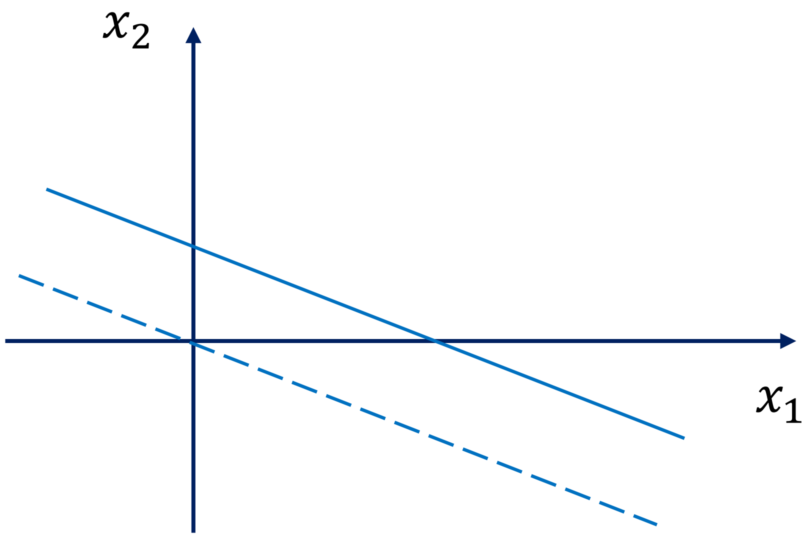
Hyperplane

- Margin M: two lines parallel to the hyperplane

- Classification via [Hyperplane with Margin]

$$\forall i \; s.t. \, y_i = +1, \;\; \vec{x}^{(i)} \cdot \vec{\beta} + \beta_0 \geq M$$

$$\forall i \; s.t. \, y_i = -1, \;\; \vec{x}^{(i)} \cdot \vec{\beta} + \beta_0 \leq -M$$



Data Separable

- Solution to the optimization problem: $\max_{\beta_0,\beta_1,\ldots,\beta_p,M} M$

$$\text{s.t.} \quad \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots + \beta_p x_p^{(i)}) \geq M$$

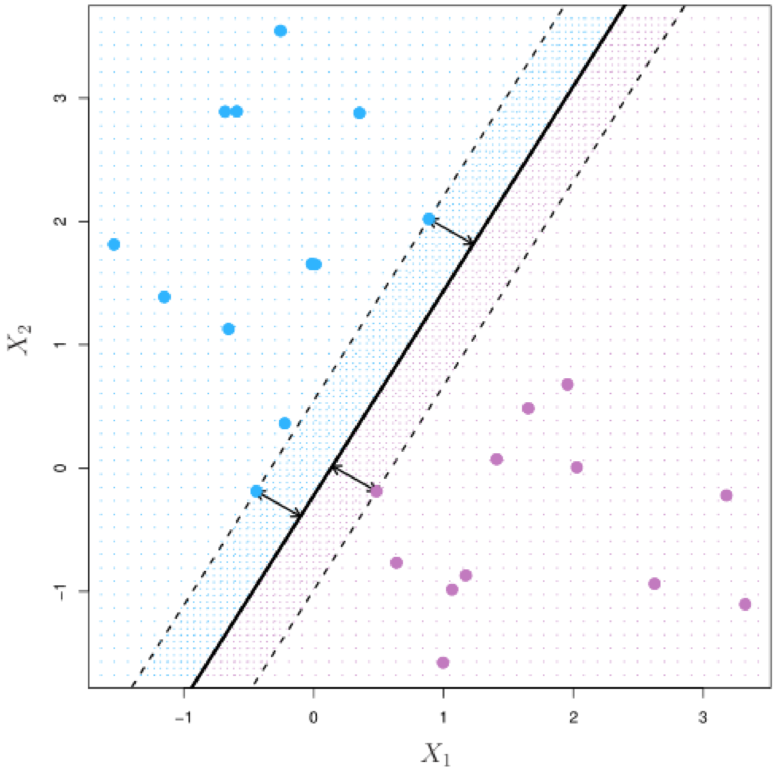
Note: Given $(\beta_0, \vec{\beta})$, hyperplane is fixed → Max M can be easily obtained

- Expand the two margin lines until one touches a data point

Thus, just search through the space of all $(\beta_0, \vec{\beta})$ → Find the optimal M

Data Separable

- Support vectors:
  - With respect to the optimal hyperplane, points that lie on the margin lines



How many support vectors are there?

Data Non-Separable

- Solution to the optimization problem $\max_{\beta_0,\beta_1,\ldots,\beta_p,\xi_i,M} M$

$$\text{s.t.} \quad \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots + \beta_p x_p^{(i)}) \geq M(1 - \xi_i)$$

$$\xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq C$$

where ξ_i 's are the slack variables, C is a nonnegative tuning parameter

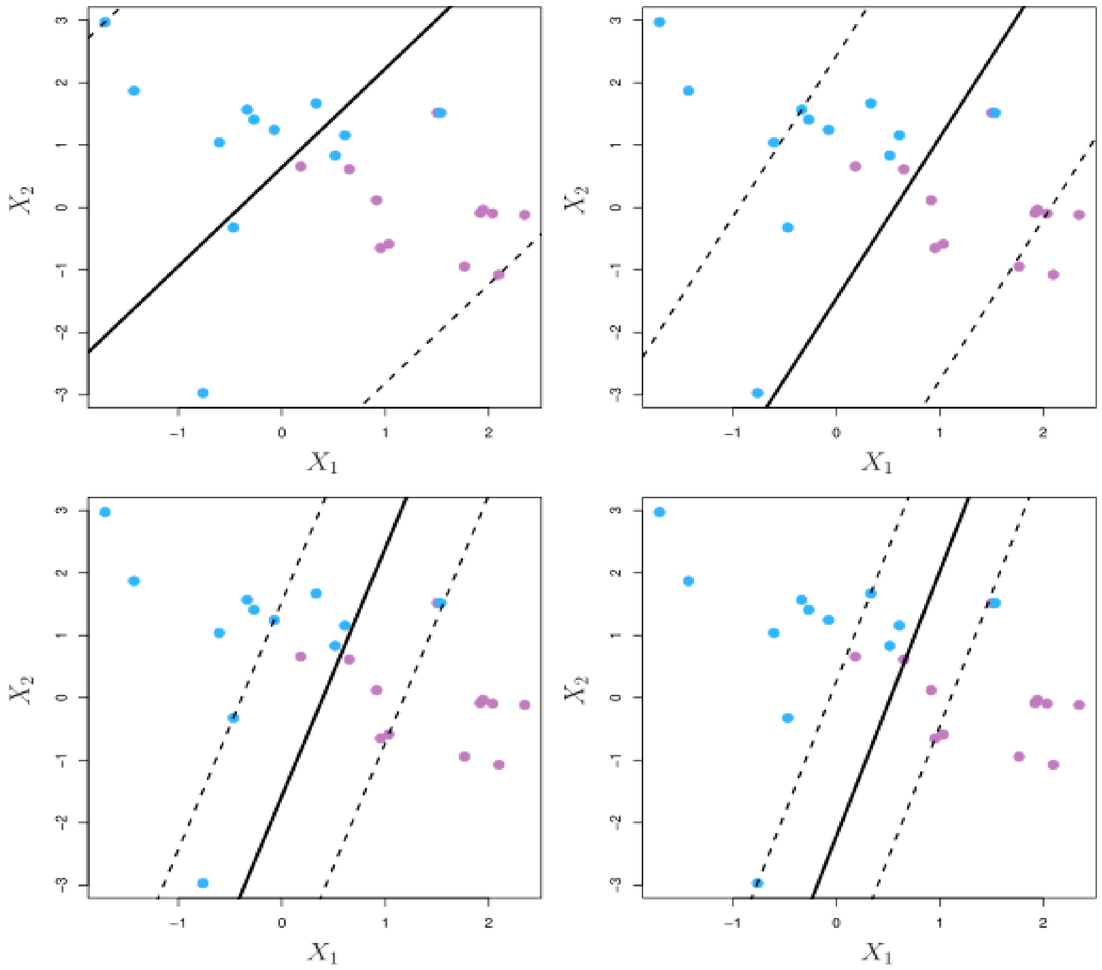
- Input: [N by p] X (training feature matrix), [N by 1] Y (training label vector)

- Optimization problem can be solved if N and p are both NOT too large

The role of C

- C determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate.

The role of *C*: Bias-variance Tradeoff

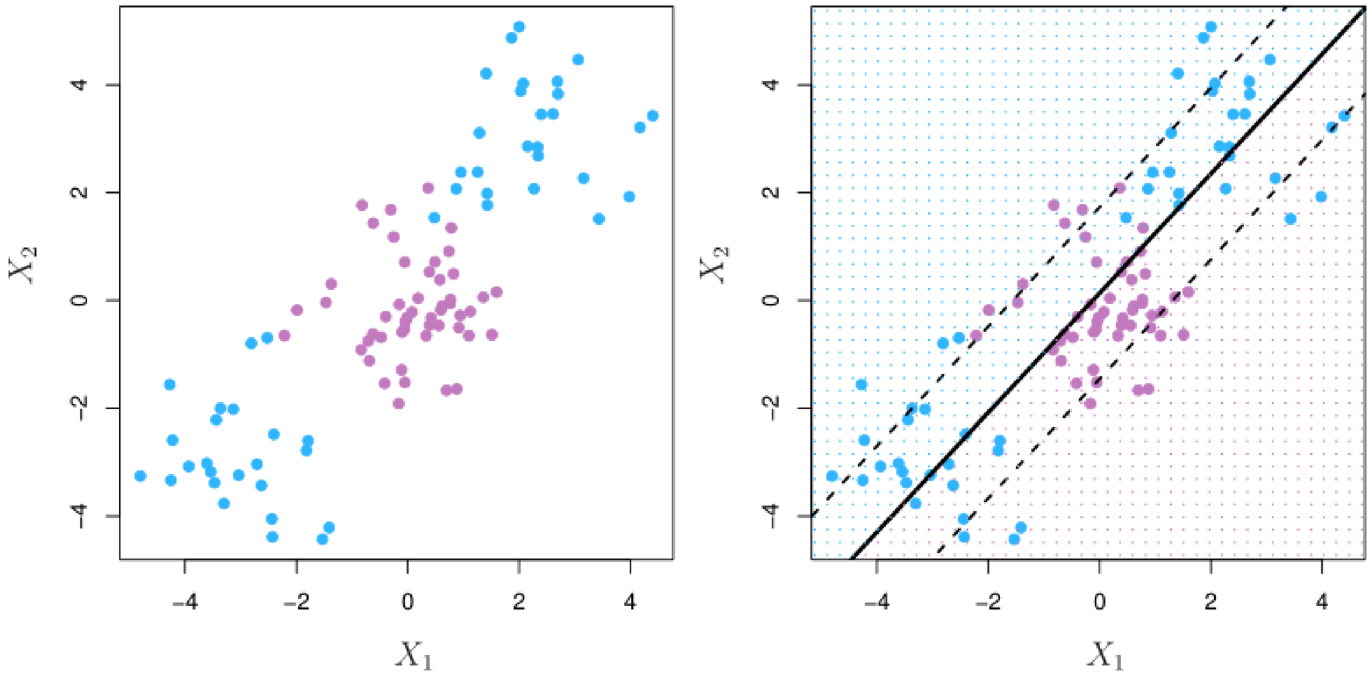


- Which figure has the largest C?

Nonlinear SVM & Kernels

Non-linear Decision Boundaries: Motivation

- Linear SVM: good approach if decision boundary linear
- In practice, many problems have non-linear boundaries



- Obvious nonlinear boundary:
  - Linear SVM (right figure) performs poorly here

Non-linear Decision Boundaries: Motivation

How to obtain nonlinear boundaries?

Non-linear Decision Boundaries

For instance, we can try the following

- Previous: x_1, x_2, …, x_p
- Now: $x_1, x_2, \dots, x_p, x_1^2, x_2^2, \dots, x_p^2$
- Optimization

$$\max_{\beta,\xi_i} M$$

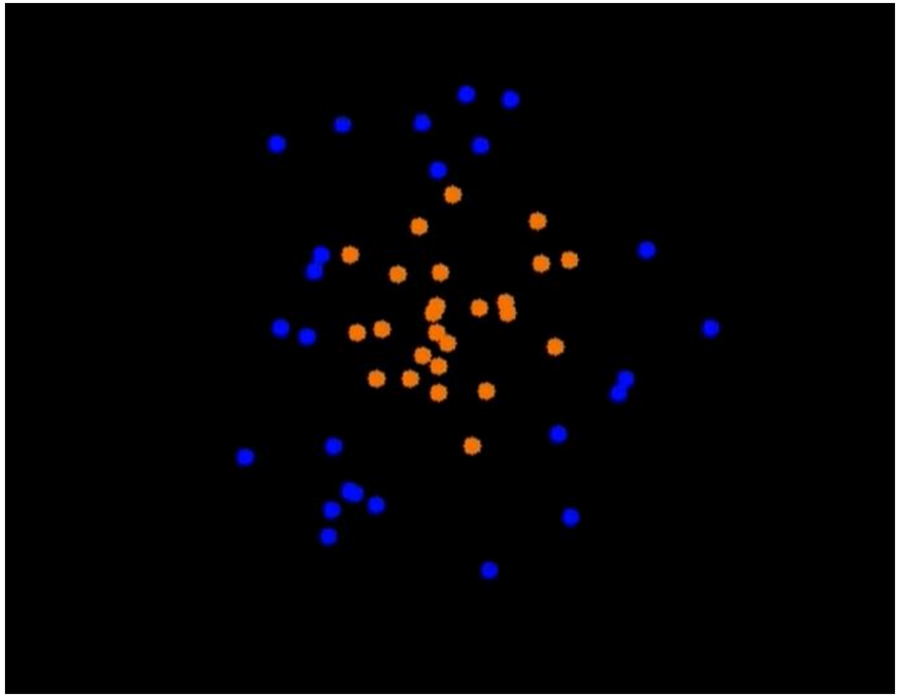
$$\sum_{j=1}^{p} \beta_{j1}^2 + \sum_{j=1}^{P} \beta_{j2}^2 = 1$$

$$y_i(\beta_0 + \sum_{j=1}^{P} \beta_{j1} x_{ij} + \sum_{j=1}^{P} \beta_{j2} x_{ij}^2) \geq M(1 - \xi_i)$$

$$\xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq C$$

- Why is this a nonlinear boundary?

Non-linear Decision Boundaries



Source: https://www.youtube.com/watch?v=9NrALgHFwTo

Non-linear Decision Boundaries

- We tried: ***p* features**: x_1, x_2, …, x_p → **2*p* features**: $x_1, x_2, \ldots, x_p, x_1^2, x_2^2, \ldots, x_p^2$
- May also want to include
  - higher-order polynomial terms / interaction terms e.g. $x_1 x_2$ / other funcs
- Problem: too many predictors
  - E.g. Include up to deg-d polynomials: from p features to O(p^d) features
  - Quickly become computational unmanageable
- We will show how to: <u>enlarge feature space</u> & <u>computationally efficient</u>

Kernel Trick Preliminaries

- ϕ: p-dim → q-dim (q > p)
  - $x^{(i)} \rightarrow \phi(x^{(i)})$, i.e. Ø transforms $x^{(i)}$ to the enlarged feature space, $i = 1,2,\dots,N$
  - E.g. (p = 2, q = 3): $x^{(i)} = (x_1^{(i)}, x_2^{(i)})$, $\phi((x_1^{(i)}, x_2^{(i)})) = \left(x_1^{(i)}, x_2^{(i)}, (x_1^{(i)})^2 + (x_2^{(i)})^2\right)$
- Kernel K: p-dim * p-dim → 1-dim
  - $K\left(x^{(i)}, x^{(j)}\right) = \phi(x^{(i)}) \cdot \phi(x^{(j)})$
- Ø is our enlarged feature space, i.e. has dim q > p. (In fact, it can be ∞-dimensional)
- This seems crazy, ↑dim from p to ∞ (p to p^d already NOT manageable for d large)
- Let's see how we can actually do that
  - 1. Important property of SVM optimization problem
  - 2. Kernel trick

Important property of Linear SVM

- Solution to the optimization problem

$$\text{s.t.} \quad \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots + \beta_p x_p^{(i)}) \geq M(1 - \xi_i)$$

$$\xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq C$$

where ξ_i's are the slack variables, C is a nonnegative tuning parameter

- Important Property: solution to the above involves only $x^{(i)} \cdot x^{(j)}, \ 1 \leq i,j \leq N$

(This slide is intended to be empty)

Important property of Linear SVM

- For the enlarged feature space (possibly ∞-dim), we just need $K\left(x^{(i)}, x^{(j)}\right)$ for all pairs of $(x^{(i)}, x^{(j)})$

- Kernel trick: compute $K\left(x^{(i)}, x^{(j)}\right) = \phi(x^{(i)}) \cdot \phi(x^{(j)})$ without using $\phi(x^{(i)})$ or $\phi(x^{(j)})$

Examples (Optional)

- $\emptyset((x_1^{(i)}, x_2^{(i)})) = ((x_1^{(i)})^2, (x_2^{(i)})^2, \sqrt{2}x_1^{(i)}x_2^{(i)})$

Examples (Optional)

- $\emptyset((x_1^{(i)}, x_2^{(i)})) = (1, \left(x_1^{(i)}\right)^2, \left(x_2^{(i)}\right)^2, \sqrt{2}x_1^{(i)}, \sqrt{2}x_2^{(i)}, \sqrt{2}x_1^{(i)}x_2^{(i)})$

Examples (Optional)

- $\mathrm{K}(x^{(i)}, x^{(j)}) = e^{-\gamma|x^{(i)}-x^{(j)}|^2}$

Kernel functions

Suppose u, $v \in \mathbb{R}_p$. Popular kernel functions include:

- Linear kernel: $K(u, v) = u^T v + c, c \in \mathbb{R}$ is a constant **Linear SVM**
- Polynomial kernel: $K(u, v) = (1 + u^T v)^d, d > 0$
- Gaussian kernel or radial basis function kernel: $K(u, v) = e^{-\gamma |u-v|^2}, \gamma > 0$

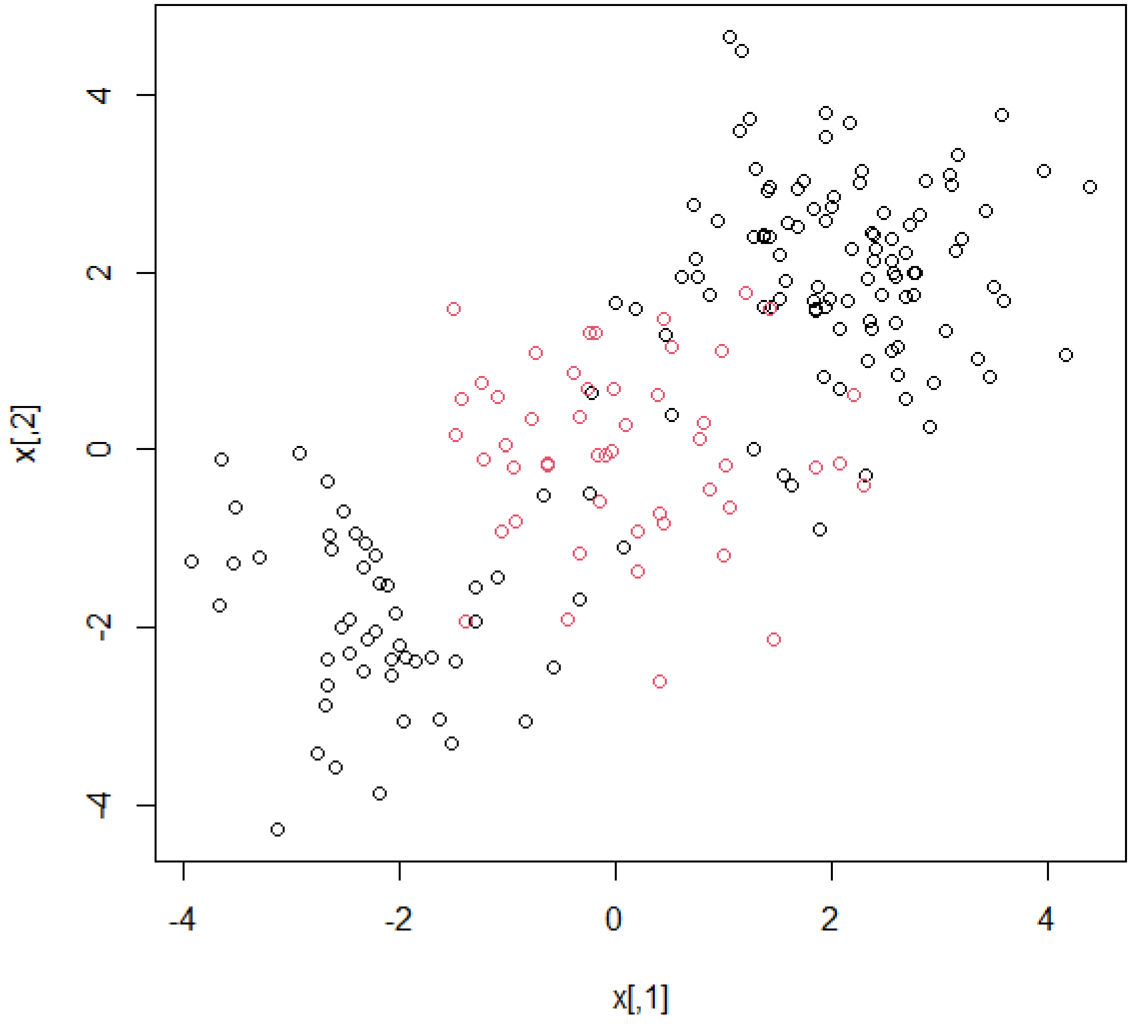
SVM Implementation

```
# Kernel SVM
library(e1071)
set.seed(1)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```

rnorm: random # from standard normal dist., then resize into 200 by 2 matrix

SVM Implementation

```
# Kernel SVM
library(e1071)
set.seed(1)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```



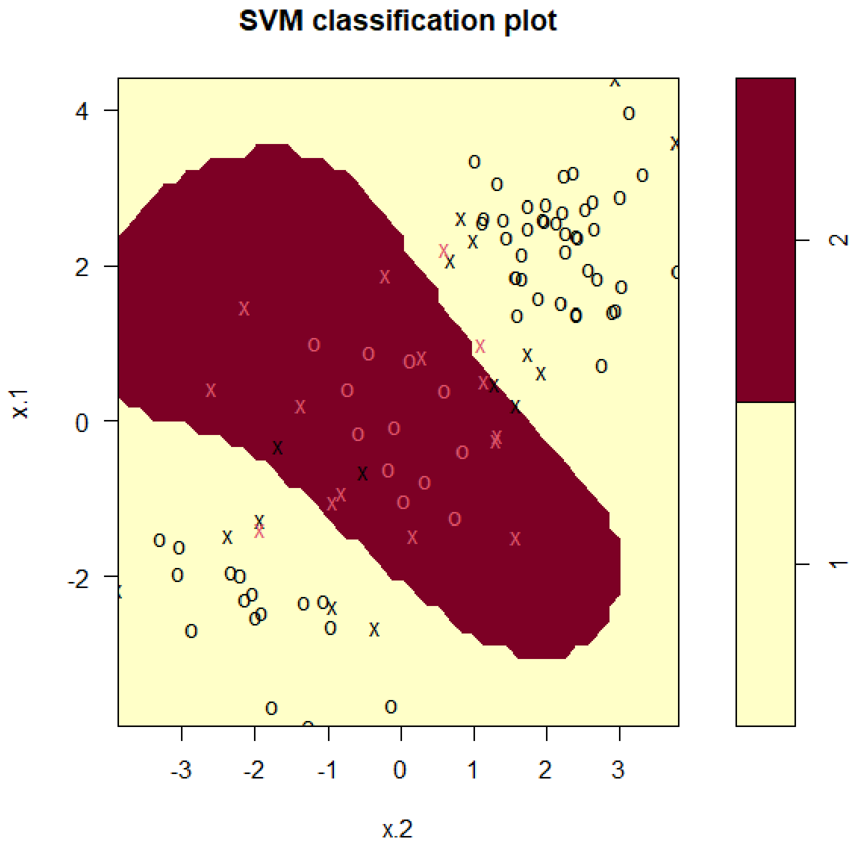
SVM Implementation

```
train=sample(200,100)

# SVM with normal C
svmfit_normalC=svm(y~., data=dat[train,], kernel="radial",  gamma=1, cost=1)
plot(svmfit_normalC, dat[train,]) # plot svm, note support vector marked as crosses
```

$$K(u, v) = e^{-\gamma \|u-v\|_2^2}$$

Radial basis kernel with gamma = 1, C = 1



SVM Implementation

$$K(u,v) = e^{-\gamma\|u-v\|_2^2}$$

Radial basis kernel with gamma = 1, C = 1

```
train=sample(200,100)

# SVM with normal C
svmfit_normalC=svm(y~., data=dat[train,], kernel="radial",  gamma=1, cost=1)
plot(svmfit_normalC, dat[train,]) # plot svm, note support vector marked as crosses
summary(svmfit_normalC)
table(true=dat[train,"y"], pred=predict(svmfit_normalC,newdata=dat[train,]))
table(true=dat[-train,"y"], pred=predict(svmfit_normalC,newdata=dat[-train,]))
```

Summary of model, train & test error

```
> summary(svmfit_normalC)

Call:
svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  31

 ( 16 15 )


Number of Classes:  2

Levels:
 1 2
```

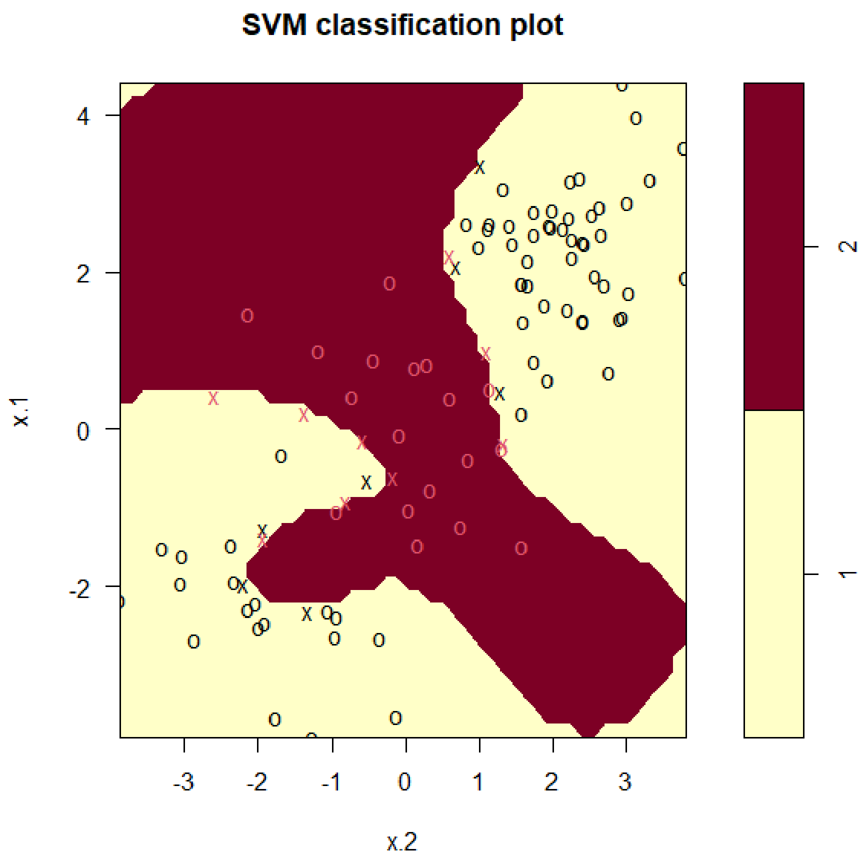
```
> table(true=dat[train,"y"], pred=predict(s
    pred
true  1  2
   1 69  4
   2  3 24
> table(true=dat[-train,"y"], pred=predict(
    pred
true  1  2
   1 67 10
   2  3 20
```

Train error: 7%

Test error: 13%

SVM Implementation

```
# SVM with large C (small margin)
svmfit_largeC=svm(y~., data=dat[train,], kernel="radial",gamma=1,cost=1e5)
plot(svmfit_largeC, dat[train,], svSymbol = "x")
```



SVM Implementation

```
# SVM with large C (small margin)
svmfit_largeC=svm(y~., data=dat[train,], kernel="radial",gamma=1,cost=1e5)
plot(svmfit_largeC, dat[train,], svSymbol = "x")
summary(svmfit_largeC)
table(true=dat[train,"y"], pred=predict(svmfit_largeC,newdata=dat[train,]))
table(true=dat[-train,"y"], pred=predict(svmfit_largeC,newdata=dat[-train,]))
```

```
> summary(svmfit_largeC)

Call:
svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1e+05)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1e+05

Number of Support Vectors:  16

 ( 7 9 )


Number of Classes:  2

Levels:
 1 2
```

Perform perfectly on the training set, poorly on the test set (overfit)

```
> table(true=dat[train,"y"], pr
     pred
true  1  2
   1 73  0
   2  0 27
> table(true=dat[-train,"y"], p
     pred
true  1  2
   1 57 20
   2  5 18
```

SVM Implementation

```
## Tune parameters
set.seed(1)
tune.out=tune(svm, y~., data=dat[train,], kernel="radial",
              ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
summary(tune.out)
```

Note: tune() performs 10-fold CV

```
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.07

- Detailed performance results:
    cost gamma error dispersion
1  1e-01   0.5  0.26 0.15776213
2  1e+00   0.5  0.07 0.08232726
3  1e+01   0.5  0.07 0.08232726
4  1e+02   0.5  0.14 0.15055453
5  1e+03   0.5  0.11 0.07378648
```

CV error of best model: 7%

SVM Implementation

```
library(dplyr)
arrange(tune.out$performances,error)

table(true=dat[-train,"y"], pred=predict(tune.out$best.model,newdata=dat[-train,]))
```

```
> arrange(tune.out$performances,error)
    cost gamma error dispersion
1  1e+00   0.5  0.07 0.08232726
2  1e+01   0.5  0.07 0.08232726
3  1e+00   1.0  0.07 0.08232726
4  1e+00   2.0  0.07 0.08232726
5  1e+00   3.0  0.07 0.08232726
6  1e+00   4.0  0.07 0.08232726
7  1e+01   3.0  0.08 0.07888106
8  1e+01   1.0  0.09 0.07378648
9  1e+01   4.0  0.09 0.07378648
10 1e+03   0.5  0.11 0.07378648
```

Test set performance

```
    pred
true  1  2
   1 67 10
   2  2 21
```

Test set error of best model: 12%

SVM Implementation

```r
# ROC Curves

library(ROCR)
rocplot=function(pred, truth, ...){
  predob = prediction(pred, truth)
  perf = performance(predob, "tpr", "fpr")
  plot(perf,...)}
svmfit.opt=svm(y~., data=dat[train,], kernel="radial",gamma=0.5, cost=1,decision.values=T)
fitted=attributes(predict(svmfit.opt,dat[train,],decision.values=TRUE))$decision.values
rocplot(-fitted,dat[train,"y"],main="Training Data")
```

pred = vector of fitted values (e.g. -1.23, 1.05)
truth = vector of true labels (e.g. 1, 2)

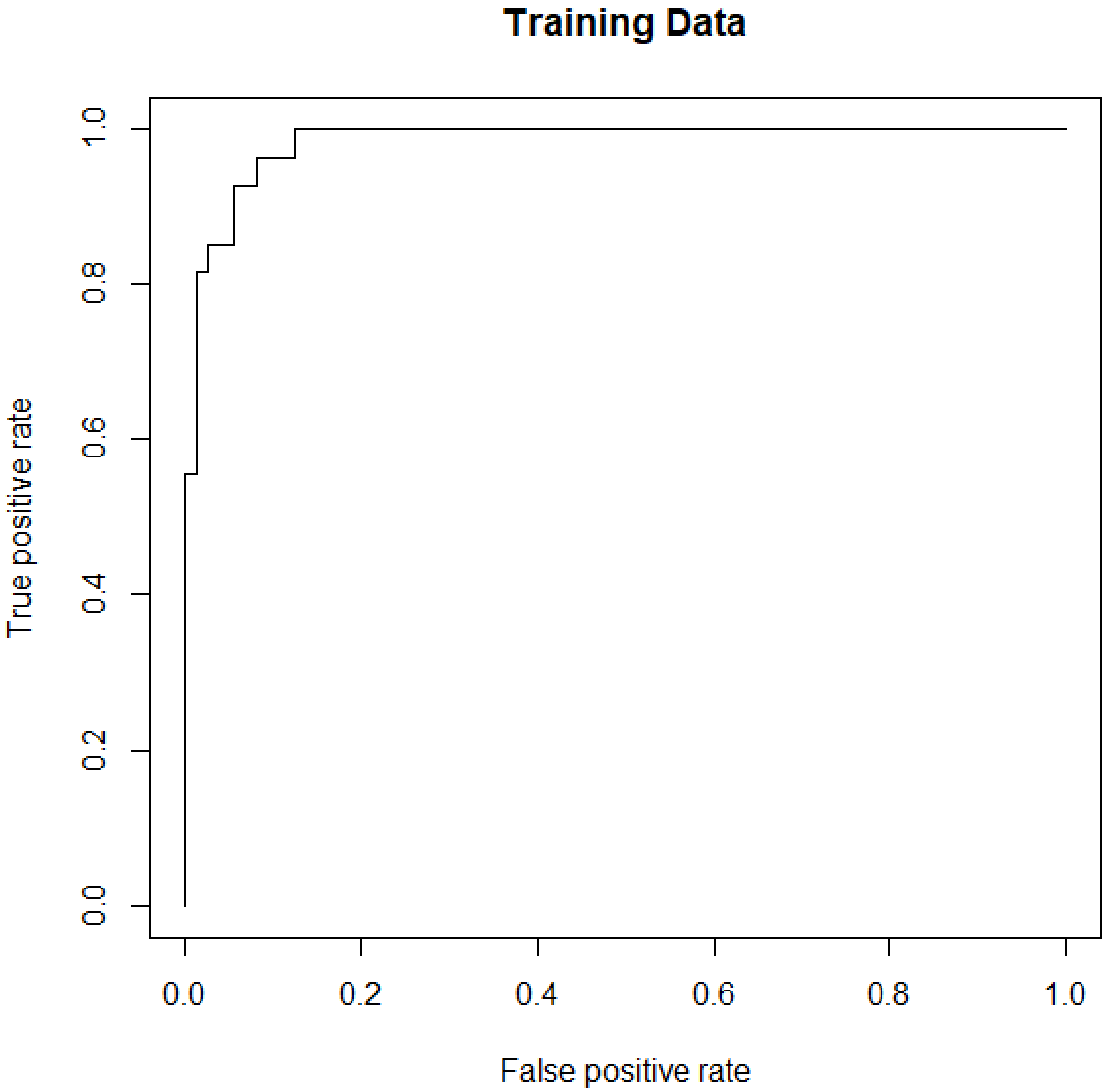
decision.values=TRUE
obtains fitted values from SVM
(e.g. -1.23, 1.05), see ISLR

decision.values=FALSE
obtains fitted class from SVM
(1 or 2)

SVM Impleme

```
# ROC Curves

library(ROCR)
rocplot=function(pre
  predob = predicti
  perf = performance
  plot(perf,...)}
svmfit.opt=svm(y~.,
fitted=attributes(pr
rocplot(-fitted,dat
```



lues (e.g. -1.23, 1.05)

els (e.g. 1, 2)

```
st=1,decision.values=T)
UE))$decision.values
```

=TRUE

alues