Big Data Analytics

Session 8

Support Vector Machines

So far

- Classifiers
  - Logistic regression
  - Decision trees
  - Ensemble learning: Bagging and Random Forests

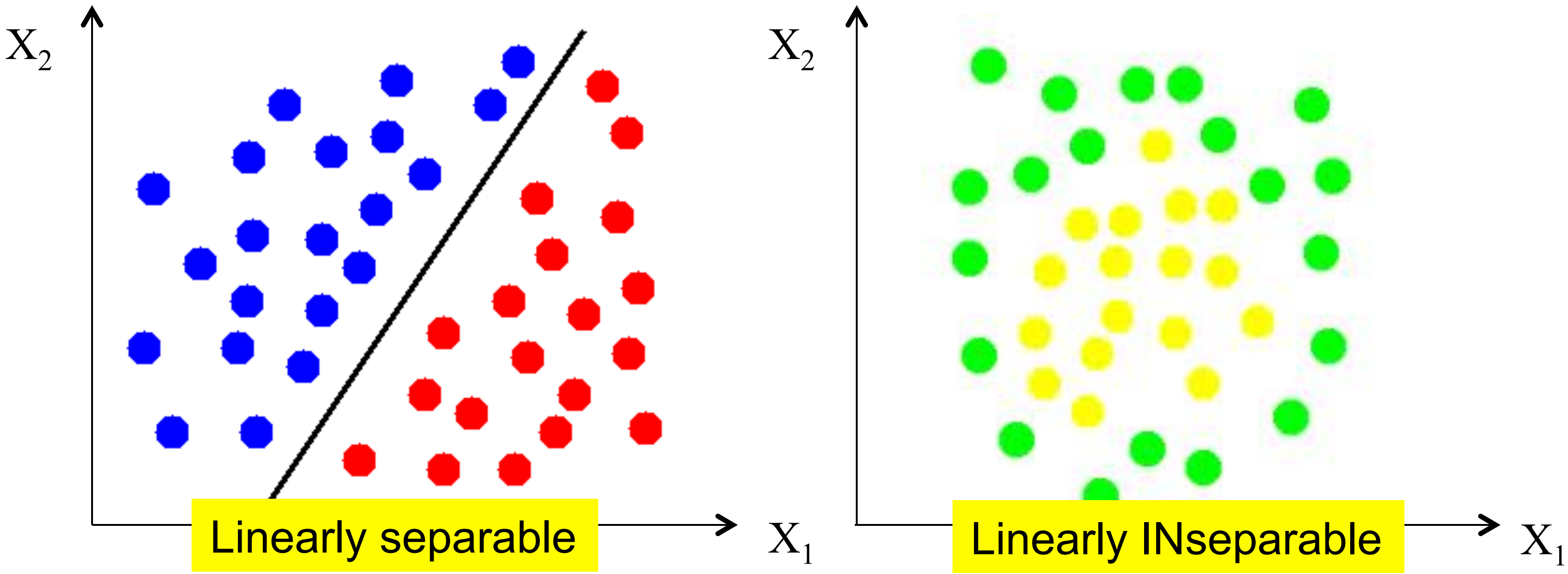
  - SVM: Support Vector Machines
    - Developed in 1990s
    - Perform well on a variety of settings
    - Often considered one of the best "out of the box" classifiers

Outline

- Maximal Margin Classifier
- The Support Vector Classifier
- (A glance at) The Support Vector Machine Classifier

Linearly Separable Classes

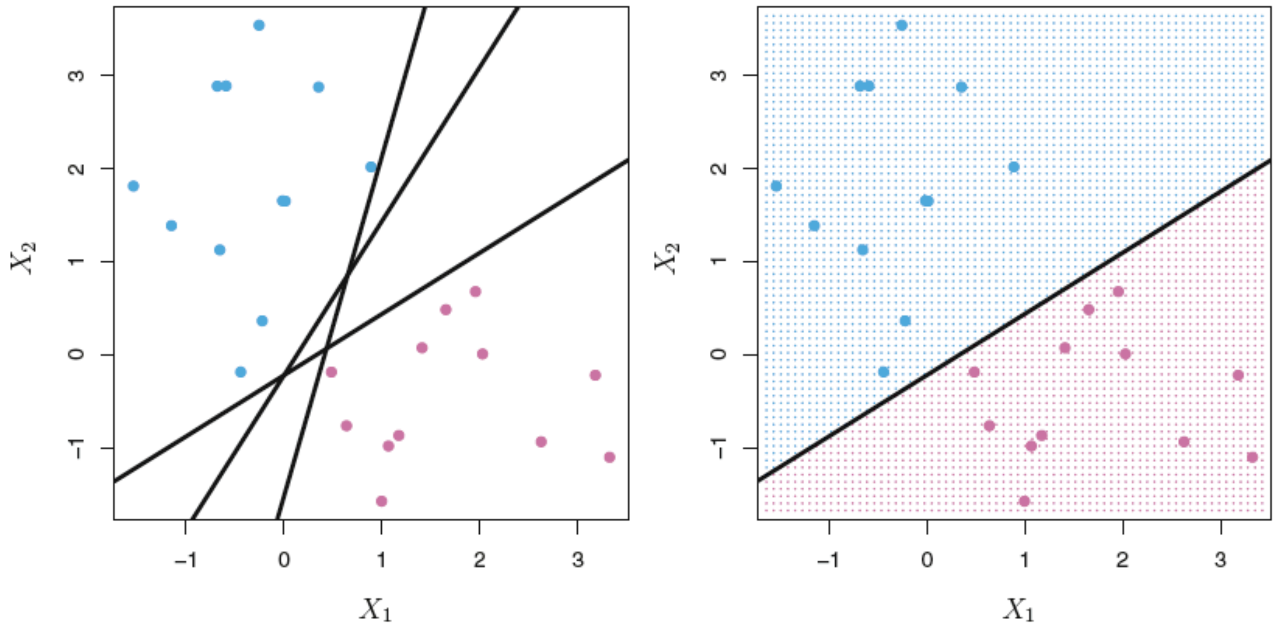
- Imagine a situation where you have a two-class classification problem with two predictors X_1 and X_2.



- Suppose that the two classes are "linearly separable" i.e. one can draw a straight line in which all points on one side belong to the first class and points on the other side to the second class.

Linearly Separable Classes

- Then a natural approach is to find the straight line that gives the biggest separation between the classes i.e. the points are as far from the line as possible.



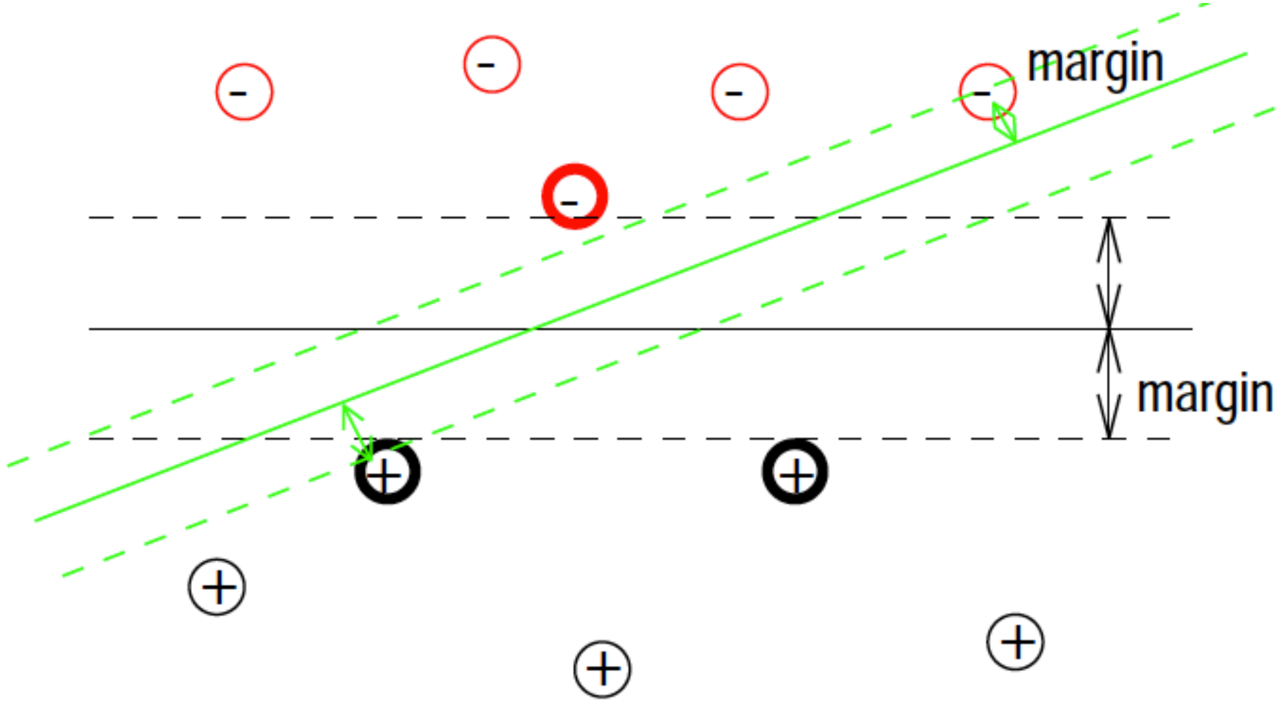
Recall:
in linear regression
Least squares line

The one with the least residual sum of squares

➔ This is the basic idea of a maximal margin classifier.

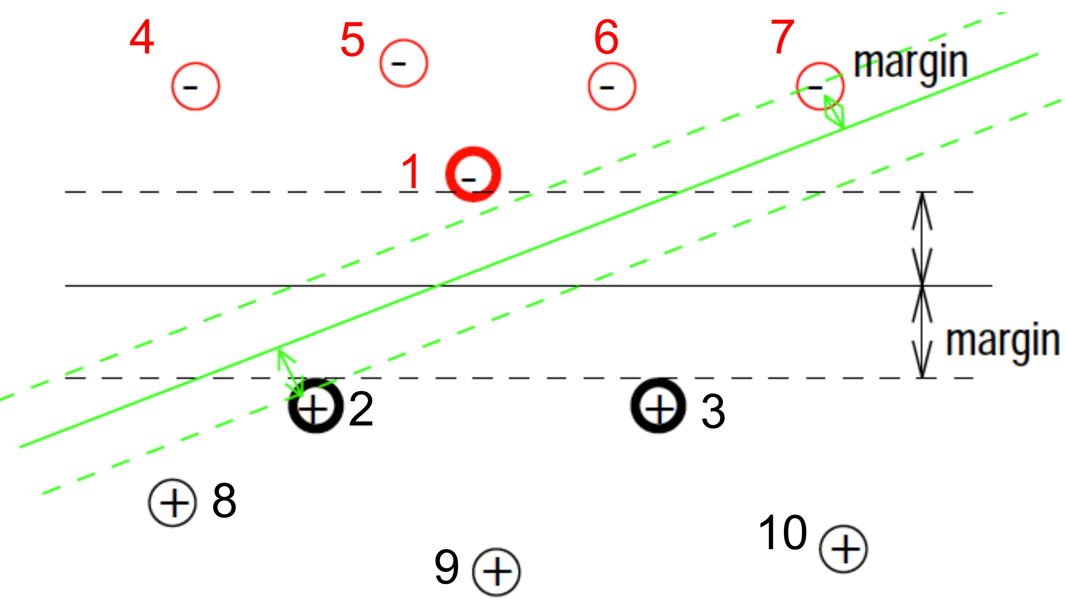
Maximal Margin Line

- Margin: the minimal (perpendicular) distance from all the observations to the separation line
- Maximal margin line: the line for which the margin is largest
- We can then use maximal margin line to classify a test observation
  - The classification of a point depends on which side of the line it falls on.



Support Vectors

- Support vectors: observations 1,2,3
  - They are on the margin
  - They are vectors (here 2-dimensional)
  - They support the maximal margin line
    - If these three points were moved, then the maximal margin line would move
  - The maximal margin line depends only on support vectors



	On the correct side of the margin	On the margin
Left	4,5,6,7 / 8,9,10	1 / 2,3

The maximal margin classifier depends only on support vectors

More Than Two Predictors

- This idea works just as well with more than two predictors.
- For example, with three predictors you want to find the plane that produces the largest separation between the classes.
- With more than three dimensions it becomes hard to visualise a plane but it still exists.
- In general they are called *hyper-planes*.
  - Two predictors: a line
  - Three predictors: a plane
  - More than three predictors: a hyper-plane

  → So we are looking for the maximal margin hyper-planes as maximal margin classifiers.

Outline

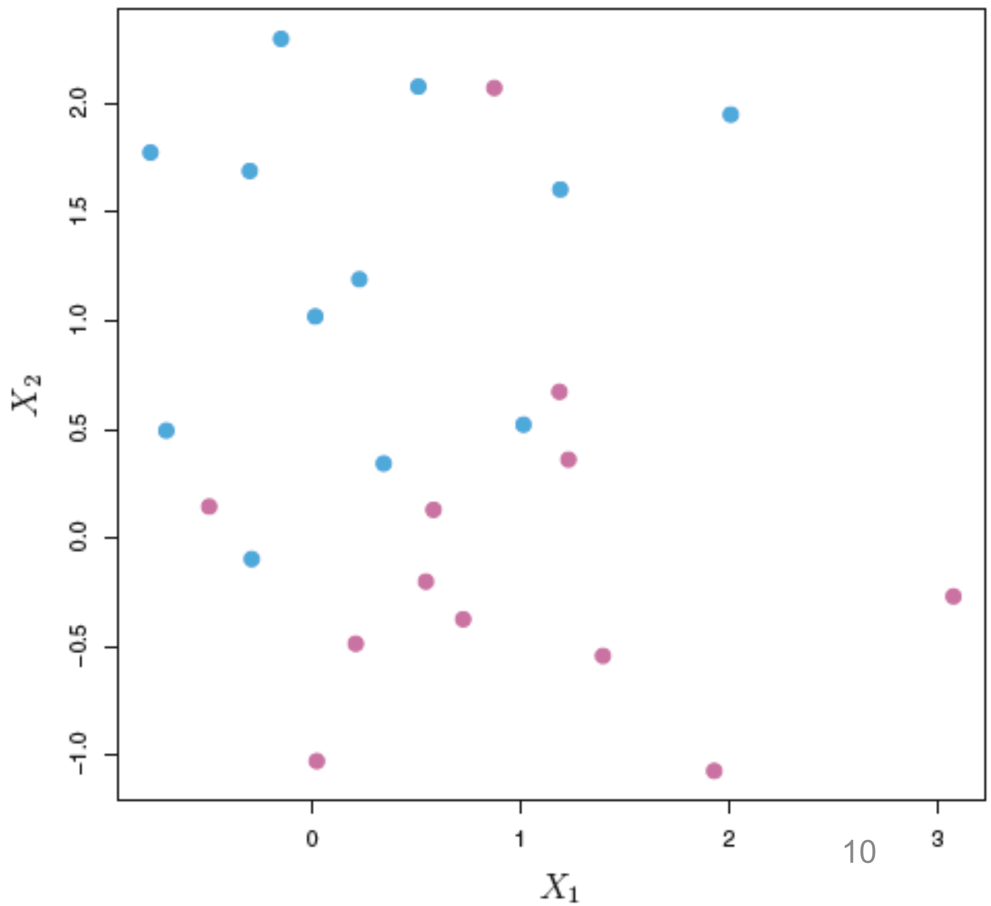
- Maximal Margin Classifier
- The Support Vector Classifier
- The Support Vector Machine Classifier

Why Maximal Margin Classifiers Are Not Ideal?

- Reason One:
  - Maximal margin hyperplanes may not exist. ➔ linearly inseparable classes

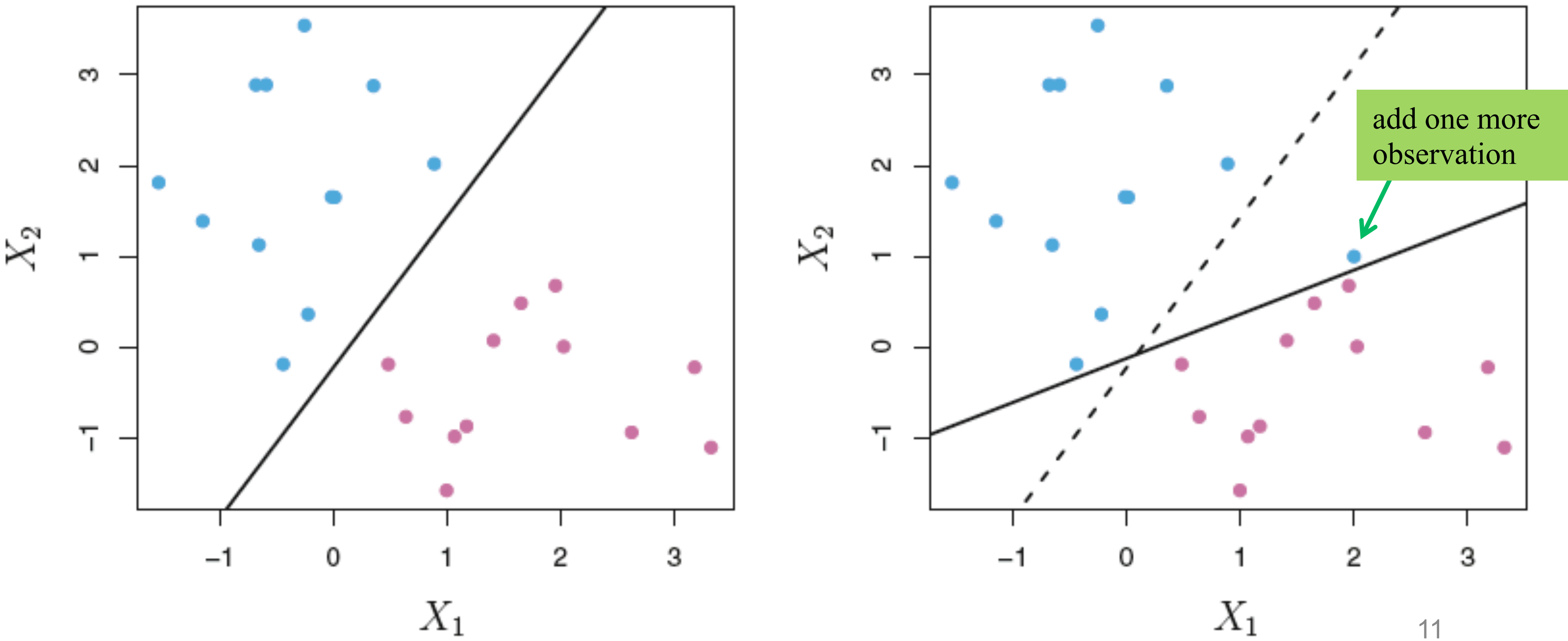
In practice it is not usually possible to find a hyper-plane that perfectly separates two classes.

In other words, for any straight line I draw there will always be at least some points on the wrong side of the line.



Why Maximal Margin Classifiers Are Not Ideal?

- Reason Two:
  - Even if maximal margin hyperplanes exist, they are extremely sensitive to a change in a single observation. → easy to overfit



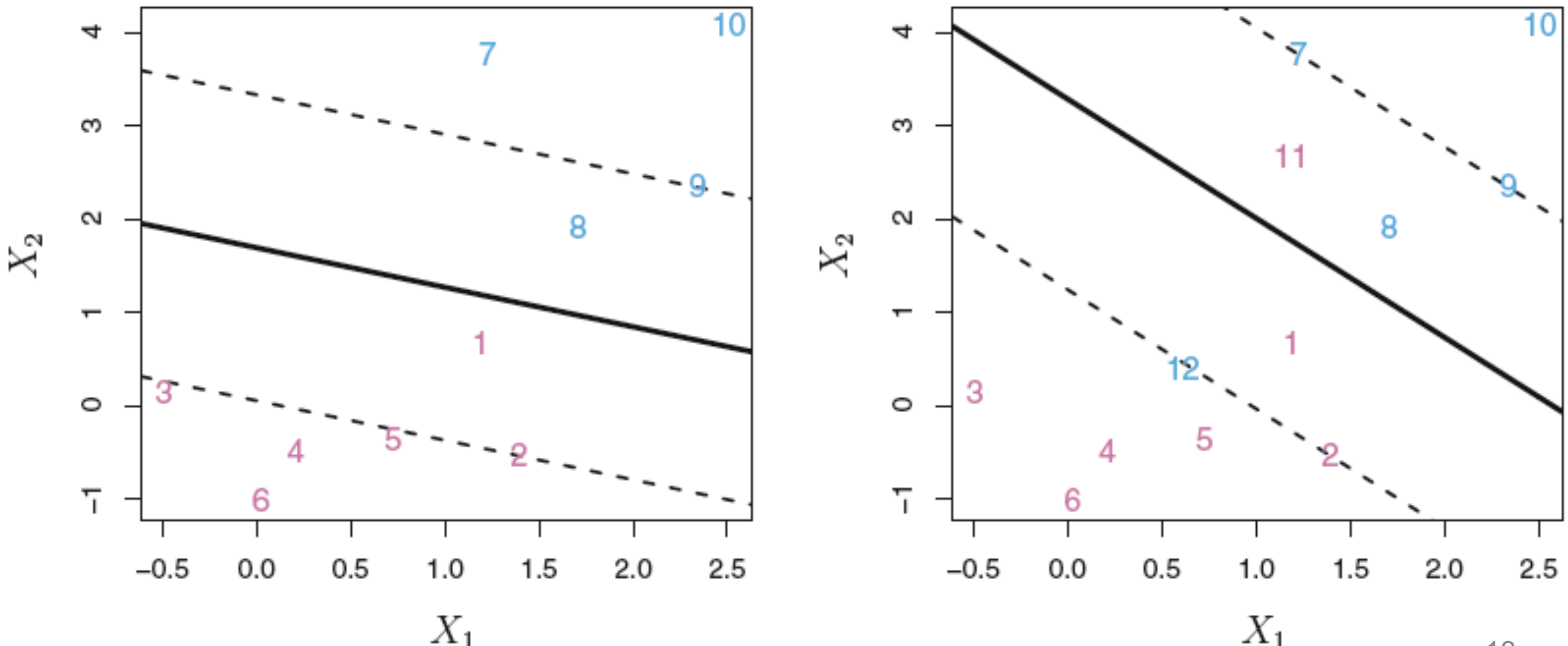
Support Vector Classifiers (SVC)

- SVCs are based on a hyperplane that does not perfectly separate the two classes, in the interest of
  - Greater robustness to individual observations, and
  - Better classification of most of the training observations.

- Soft margin
  - We allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane.

SVC Examples

	On the correct side of the margin	On the margin	On the wrong side of the margin	On the wrong side of the hyperplane
Left				
Right				

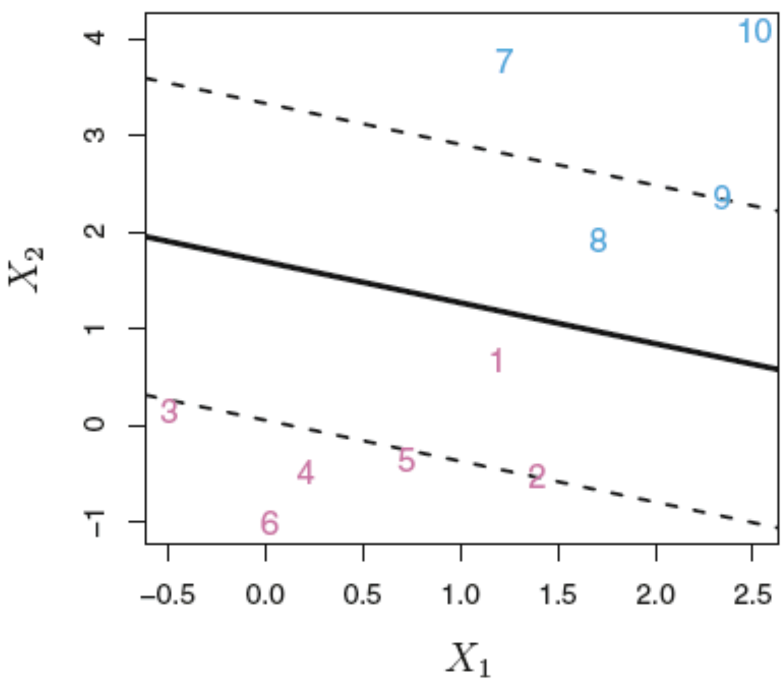


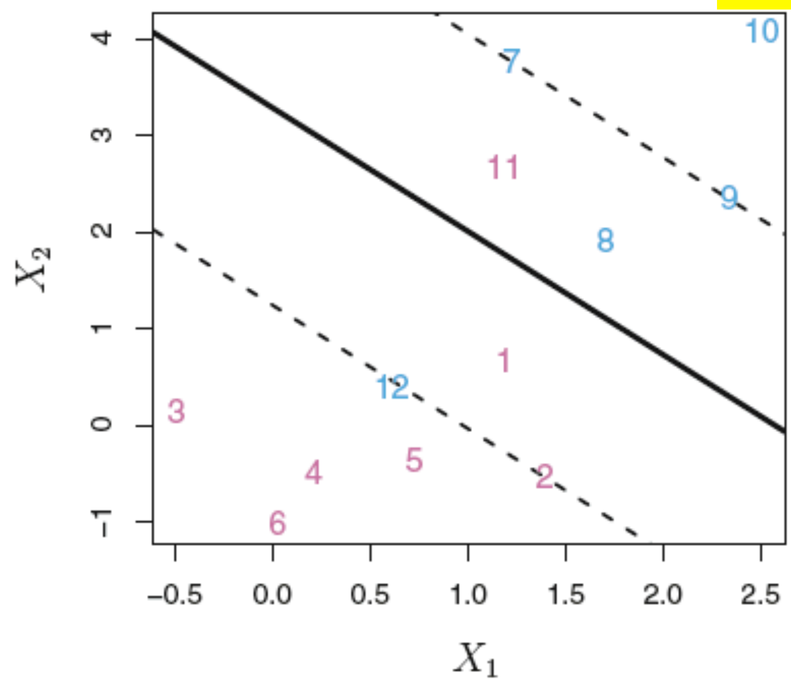
SVC Examples

The SVC depends only on ***support vectors***

	On the correct side of the margin	On the margin	On the wrong side of the margin	On the wrong side of the hyperplane
Left	3,4,5,6 / 7,10	2 / 9	1 / 8	none
Right	3,4,5,6 / 7,10	2 / 9	1 / 8	11 / 12

The latter two columns are not allowed in the MMC.





Cost

- A `Cost` allows us to specify the cost of a violation to the margin.

Cost

- A `Cost` allows us to specify the cost of a violation to the margin.
  - `Cost` is a tuning parameter and is generally chosen via cross validation.
    - The choice of cost is very important
    - It determines the extent to which the model underfits or overfits the data.
  - When cost is large, then
    - The margin will be narrow
    - There will be few support vectors involved in determining the hyperplane
    - Amounts to a classifier that is highly fit to the data
    - Low bias and high variance
  - When cost is small, then
    - The margin will be wide
    - Many support vectors will be involved in determining the hyperplane
    - Amounts to fitting the data less hard
    - High bias and low variance

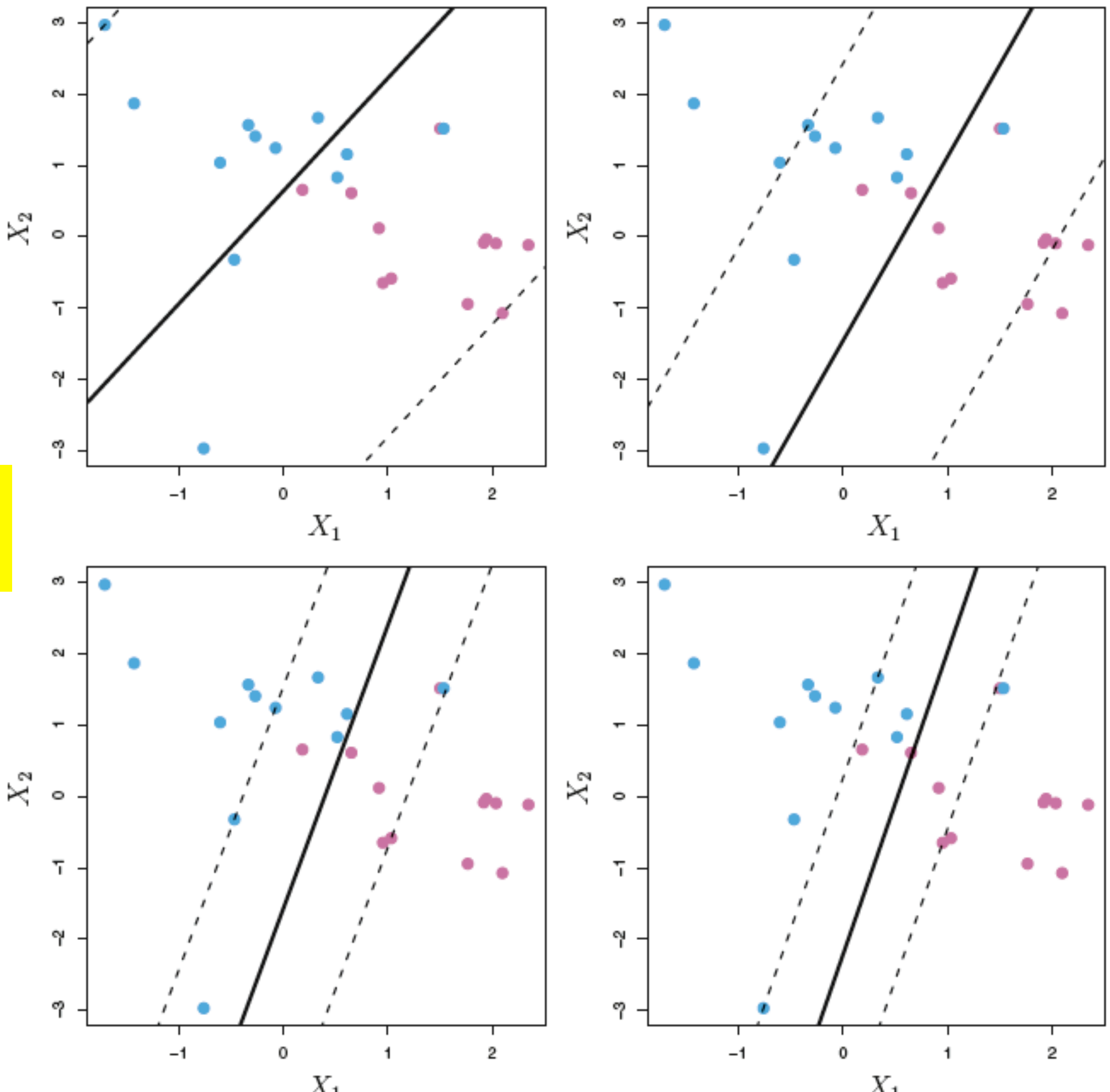
1) Margin narrow or wide?

2) Fewer or more support vectors?

3) Classifier highly fit to the data or not?

4) Bias? Variance?

Cost Examples



cost=0.01

cost=0.1

cost=1

cost=10

Which has the least cost?

Which has low variance, but potentially high bias?

Some Remarks

- In the book, “budget C” is used to explain the concept rather than “cost”. Budget and cost are dual.
  - The higher the budget is, the smaller the cost is.
  - The lower the budget is, the bigger the cost is.

- Which points should influence optimality?
  - All points
    - Linear regression
    - Naïve Bayes
    - Linear discriminant analysis
  - Only “difficult points” close to decision boundary
    - Support vector machines
    - Logistic regression (kind of) [See section 9.5 for more details]

Support Vector Classifier

- To demonstrate the SVC (and SVM), we use
  - `e1071` library or
  - `LiblineaR` library (useful for very large linear problems)

- Use `svm()` function to fit a support vector classifier/machine
  - With `kernel="linear"` to fit a SVC, otherwise a SVM
  - With `cost` argument: specify the cost of a violation to the margin
    - `cost` is small: wide margins
      - many support vectors will be on the margin or will violate the margin
    - `cost` is large: narrow margins
      - Few support vectors will be on the margin or will violate the margin

A Two-Dimensional Example

```
> set.seed(1)

> x=matrix(rnorm(20*2),ncol=2)
> x

> y=c(rep(-1,10),rep(1,10))
> y
 [1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
      1  1  1  1  1  1  1  1  1  1

> x[y==1,]=x[y==1,]+1
> x
```

```
> x
```

	[,1]	[,2]
[1,]	-0.62645381	0.91897737
[2,]	0.18364332	0.78213630
[3,]	-0.83562861	0.07456498
[4,]	1.59528080	-1.98935170
[5,]	0.32950777	0.61982575
[6,]	-0.82046838	-0.05612874
[7,]	0.48742905	-0.15579551
[8,]	0.73832471	-1.47075238
[9,]	0.57578135	-0.47815006
[10,]	-0.30538839	0.41794156
[11,]	1.51178117	1.35867955
[12,]	0.38984324	-0.10278773
[13,]	-0.62124058	0.38767161
[14,]	-2.21469989	-0.05380504
[15,]	1.12493092	-1.37705956
[16,]	-0.04493361	-0.41499456
[17,]	-0.01619026	-0.39428995
[18,]	0.94383621	-0.05931340
[19,]	0.82122120	1.10002537
[20,]	0.59390132	0.76317575

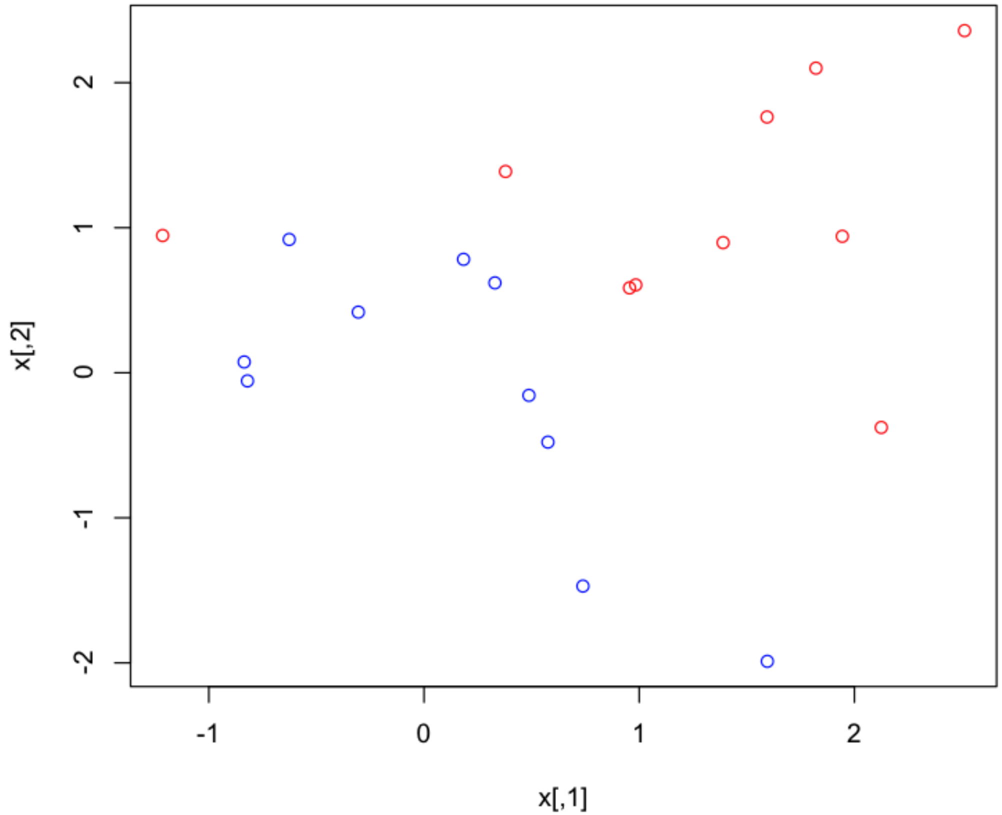
```
> x
```

	[,1]	[,2]
[1,]	-0.6264538	0.91897737
[2,]	0.1836433	0.78213630
[3,]	-0.8356286	0.07456498
[4,]	1.5952808	-1.98935170
[5,]	0.3295078	0.61982575
[6,]	-0.8204684	-0.05612874
[7,]	0.4874291	-0.15579551
[8,]	0.7383247	-1.47075238
[9,]	0.5757814	-0.47815006
[10,]	-0.3053884	0.41794156
[11,]	2.5117812	2.35867955
[12,]	1.3898432	0.89721227
[13,]	0.3787594	1.38767161
[14,]	-1.2146999	0.94619496
[15,]	2.1249309	-0.37705956
[16,]	0.9550664	0.58500544
[17,]	0.9838097	0.60571005
[18,]	1.9438362	0.94068660
[19,]	1.8212212	2.10002537
[20,]	1.5939013	1.76317575

rnorm() generates a vector of random normal variables
matrix(rnorm(20*2),ncol=2) generates a 20*2 matrix of 40 random normal variables
By default, byrow=FALSE. In other words, fill the matrix column-wise.

A Two-Dimensional Example

- Check whether the classes are linearly separable
  - `plot(x,col=(3-y))`



Not linearly separable!

> palette()
> [1] "black" "red" "green3" "blue" "cyan" "magenta" "yellow" "gray"

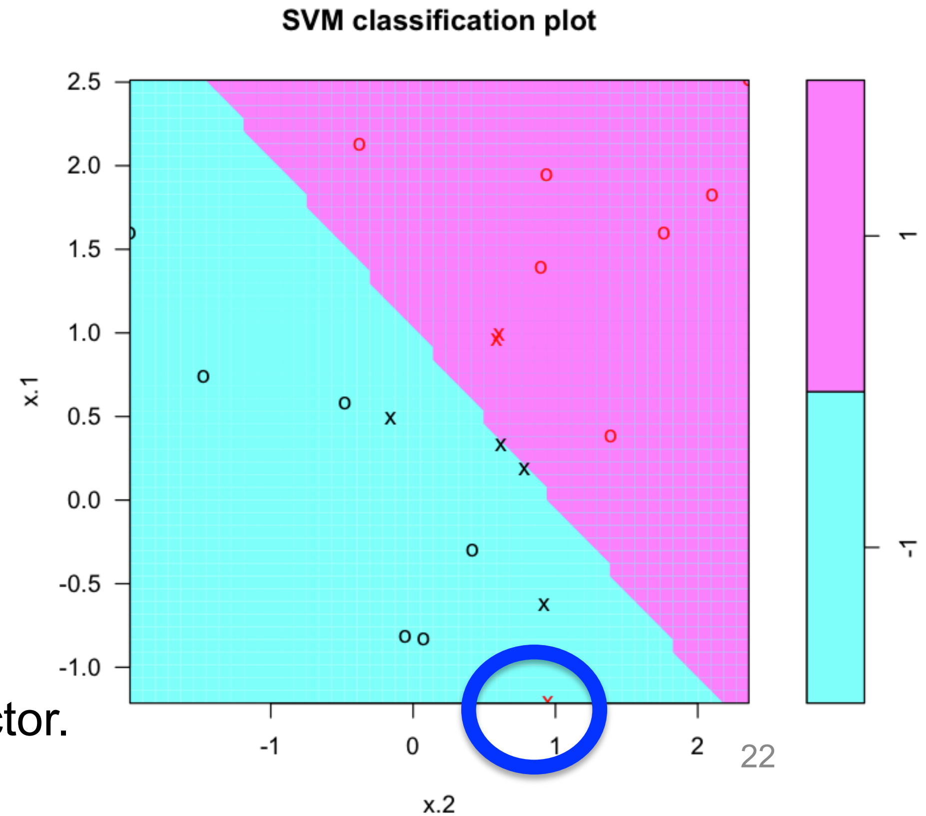
A Two-Dimensional Example

- Encode the response as a factor variable by creating a data frame:

```
> dat=data.frame(x=x,y=as.factor(y))
> library(e1071)
> svmfit=svm(y~.,data=dat,kernel="linear",cost=10,scale=FALSE)
```

- `plot(svmfit,dat)`
  - y=-1 blue; y=+1 purple
  - linear boundary
  - One misclassification
  - Support vectors: cross; remaining: circle
  - 7 support vectors:

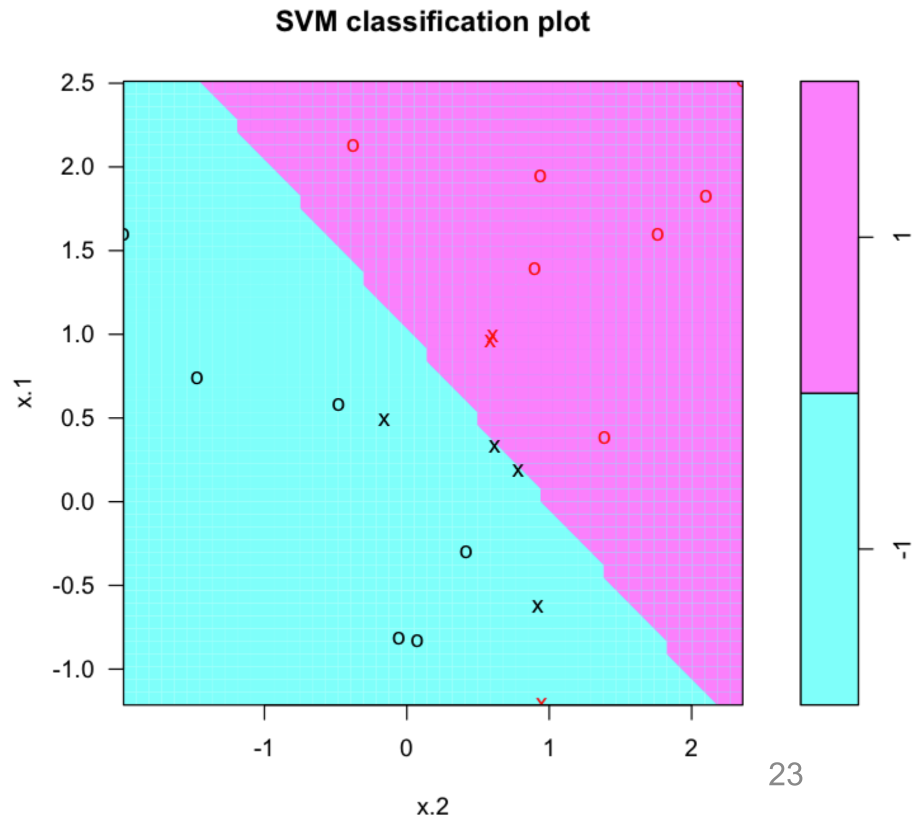
```
> svmfit$index
[1]  1  2  5  7 14 16 17
```



`as.factor` coerces its argument to a factor.

A Two-Dimensional Example

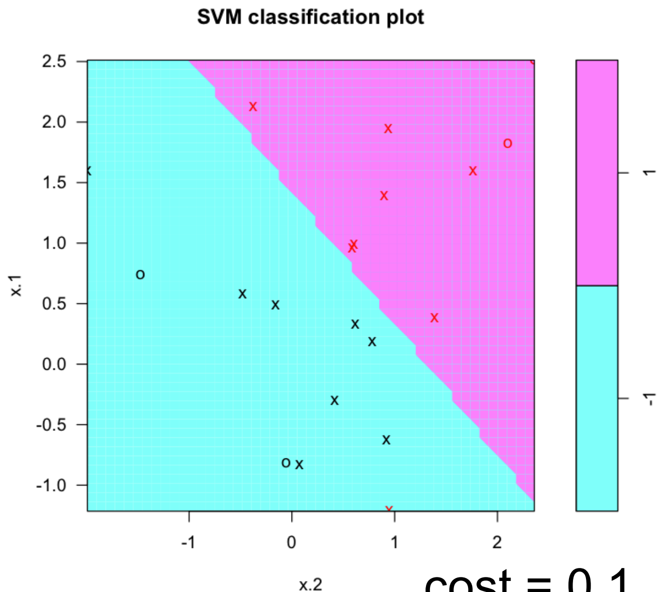
```
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5

Number of Support Vectors:  7

 ( 4 3 )

Number of Classes:  2

Levels:
 -1 1
```



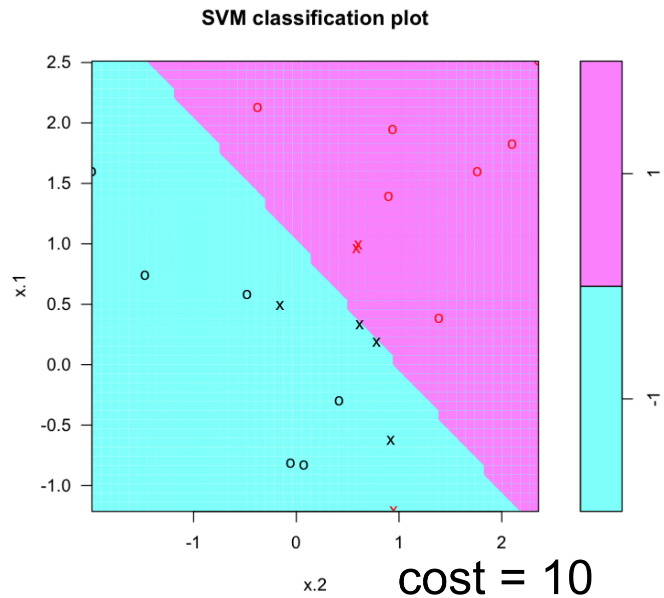
A Two-Dimensional Example

- Try a smaller cost:

```
> svmfit=svm(y~.,data=dat,kernel="linear",cost=0.1,scale=FALSE)
> plot(svmfit,dat)
> svmfit$index
 [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```



cost = 0.1



cost = 10

- Smaller cost ➔ a larger number of support vectors, a wider margin

Try Another Function

- `tune()` in `e1071` library
  - Perform 10-fold cross-validation
- Compare SVMs with a linear kernel, using a range of values of the `cost` parameter

```
> set.seed(1)
> tune.out=tune(svm,y~.,data=dat,kernel="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
> summary(tune.out)
```

```
Parameter tuning of ‘svm’:
-   sampling method:
    10-fold cross validation
-   best parameters: cost  0.1
-   best performance: 0.1

- Detailed performance results:
   cost error dispersion
1 1e-03  0.70  0.4216370
2 1e-02  0.70  0.4216370
3 1e-01  0.10  0.2108185
4 1e+00  0.15  0.2415229
5 5e+00  0.15  0.2415229
6 1e+01  0.15  0.2415229
7 1e+02  0.15  0.2415229
```

```
> bestmod=tune.out$best.model
> summary(bestmod)

Call:
best.tune(method = svm, train.x = y ~ ., data = dat,
ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),
    kernel = "linear”)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5

Number of Support Vectors:  16  ( 8 8 )

Number of Classes:  2
```

Another example of using CV to compare and select model

Predict Class Labels

- First generate a test data set

```
> xtest=matrix(rnorm(20*2),ncol=2)
> ytest=sample(c(-1,1),20,rep=TRUE)
```

#rep: Should sampling be with replacement?

```
> ytest
 [1]  1 -1 -1  1  1 -1 -1 -1  1  1  1  1
-1 -1 -1 -1  1 -1 -1  1
> xtest
              [,1]        [,2]
 [1,]  1.51178117  1.35867955
 [2,]  0.38984324 -0.10278773
 [3,] -0.62124058  0.38767161
 [4,] -2.21469989 -0.05380504
 [5,]  1.12493092 -1.37705956
 [6,] -0.04493361 -0.41499456
 [7,] -0.01619026 -0.39428995
 [8,]  0.94383621 -0.05931340
 [9,]  0.82122120  1.10002537
[10,]  0.59390132  0.76317575
[11,]  0.91897737 -0.16452360
  ……
[18,] -1.47075238  0.76853292
[19,] -0.47815006 -0.11234621
[20,]  0.41794156  0.88110773
```

```
> xtest[ytest==1,]=xtest[ytest==1,]+1
> testdat=data.frame(x=xtest,y=as.factor(ytest))
> testdat
           x.1        x.2  y
1   2.51178117  2.3586796  1
2   0.38984324 -0.1027877 -1
3  -0.62124058  0.3876716 -1
4  -1.21469989  0.9461950  1
5   2.12493092 -0.3770596  1
6  -0.04493361 -0.4149946 -1
7  -0.01619026 -0.3942900 -1
8   0.94383621 -0.0593134 -1
9   1.82122120  2.1000254  1
10  1.59390132  1.7631757  1
11  1.91897737  0.8354764  1
 ……
18 -1.47075238  0.7685329 -1
19 -0.47815006 -0.1123462 -1
20  1.41794156  1.8811077  1
```

Predict Class Labels

- Then predict the class labels of these test observations
  - First using the best model (with cost=0.1)

```
> ypred=predict(bestmod,testdat)
> table(predict=ypred,truth=testdat$y)    # build the confusion matrix
       truth
predict -1  1
     -1 11  1
     1   0  8
```

With cost = 0.1, 19 of the test observations are correctly classified.

  - What if cost=0.01?

```
> svmfit=svm(y~.,data=dat,kernel="linear",cost=.01,scale=FALSE)
> ypred=predict(svmfit,testdat)
> table(predict=ypred,truth=testdat$y)
       truth
predict -1  1
     -1 11  2
     1   0  7
```

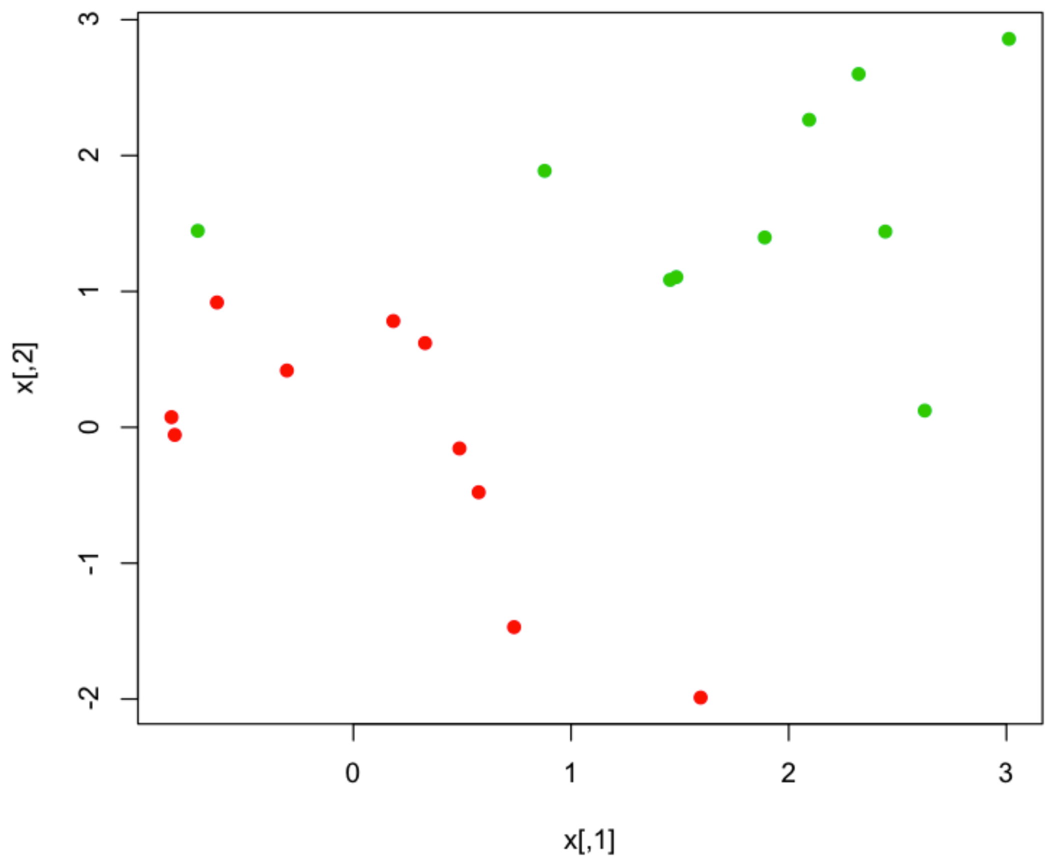
With cost = 0.01, 18 of the test observations are correctly classified.

You may try cost=1, 5, 10 or other values

A Linearly Separable Example

- First generate a linearly separable training set

```
> x=matrix(rnorm(20*2),ncol=2)
> y=c(rep(-1,10),rep(1,10))

> x[y==1,]=x[y==1,]+0.5
> plot(x,col=(y+5)/2,pch=19)
```



A Linearly Separable Example

- We fit the SVC and plot the resulting hyperplane, using a very large value of cost so that no observations are misclassified

```
> dat=data.frame(x=x,y=as.factor(y))
> svmfit=svm(y~.,data=dat,kernel="linear",cost=1e5)
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e+05
      gamma:  0.5

Number of Support Vectors:  3 ( 1 2 )

Number of Classes:  2

Levels:
 -1 1

> plot(svmfit,dat)
```

A Linearly Separable Example

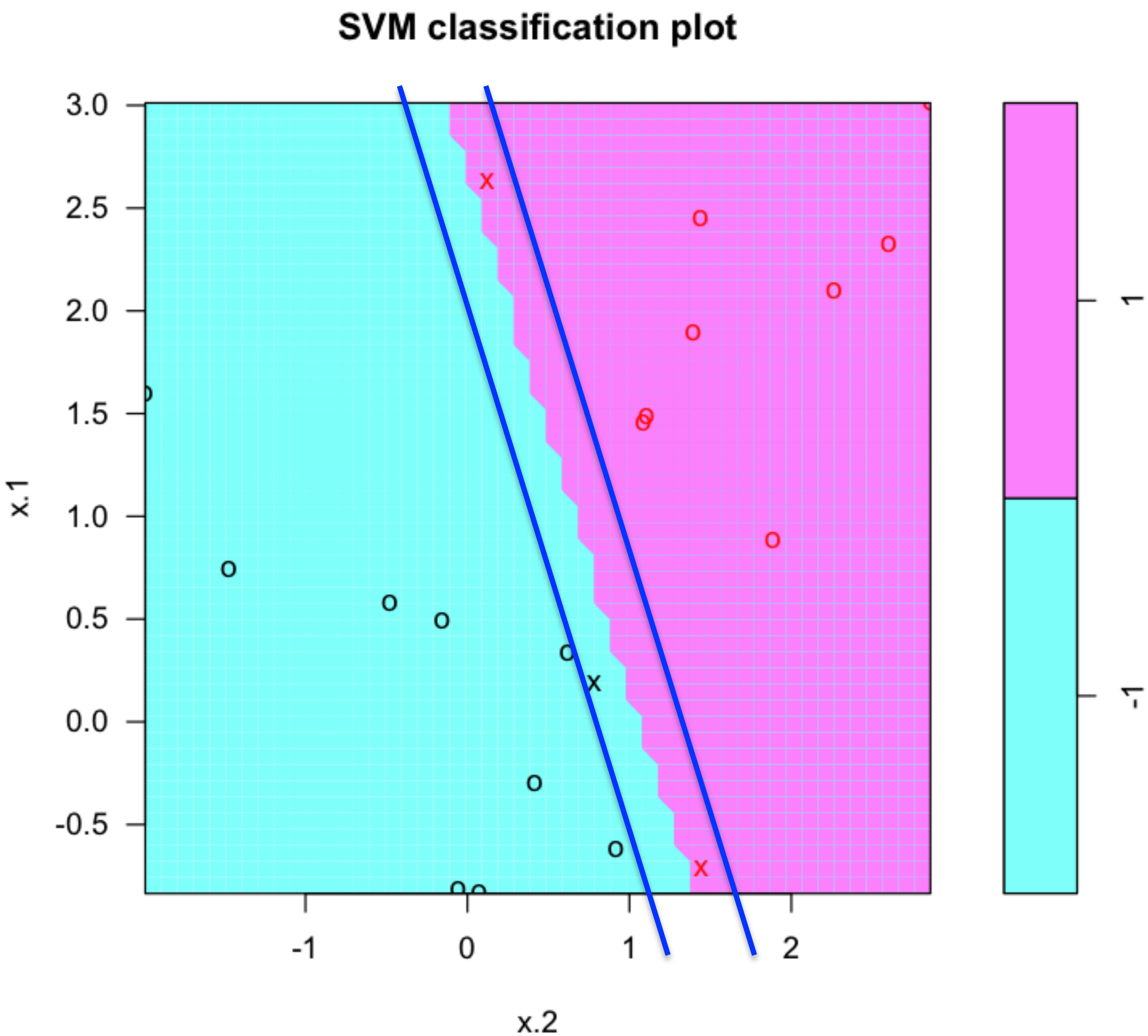
Only 3 support vectors were used.

The margin is very narrow.

However, some circle observations are very close to the decision boundary.

It seems that this model will perform poorly on test data.

Your task: generate a test dataset and calculate the test error rate.



A Linearly Separable Example

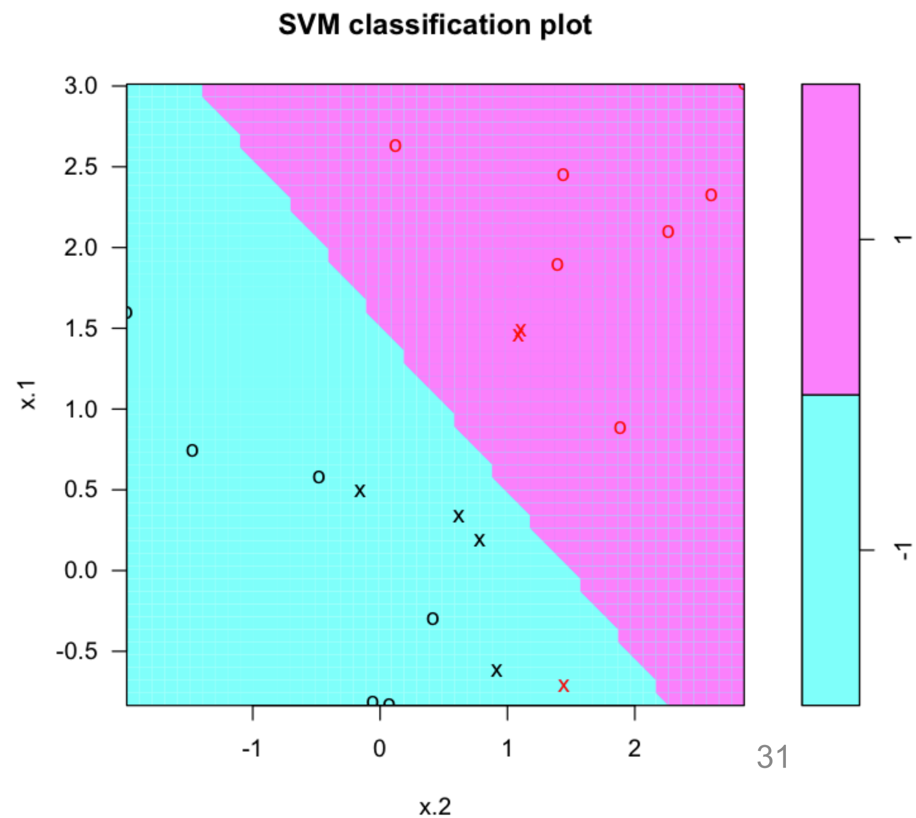
- Now try a smaller value of `cost`:

```
> svmfit=svm(y~.,data=dat,kernel="linear",cost=1)
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  7 ( 4 3 )
Number of Classes:  2
Levels:
 -1 1

> plot(svmfit,dat)
```

Misclassify one training observation, but a much wider margin and 7 support vectors

May perform better than the previous one

Your task: To use the same test dataset and calculate the test error rate. Compare the error rate with the one on the previous slide.

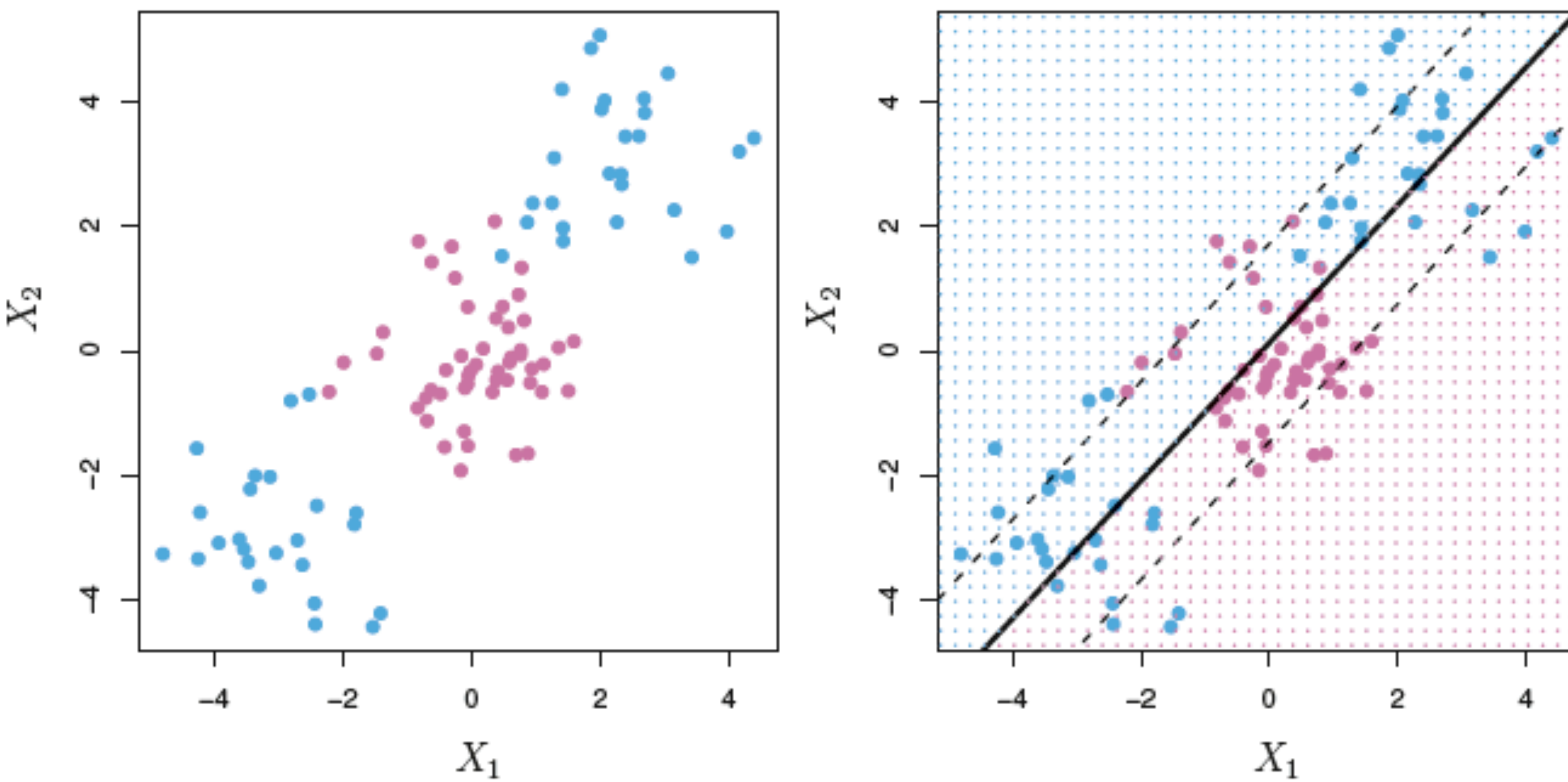


Outline

- Maximal Margin Classifier
- The Support Vector Classifier
- The Support Vector Machine Classifier

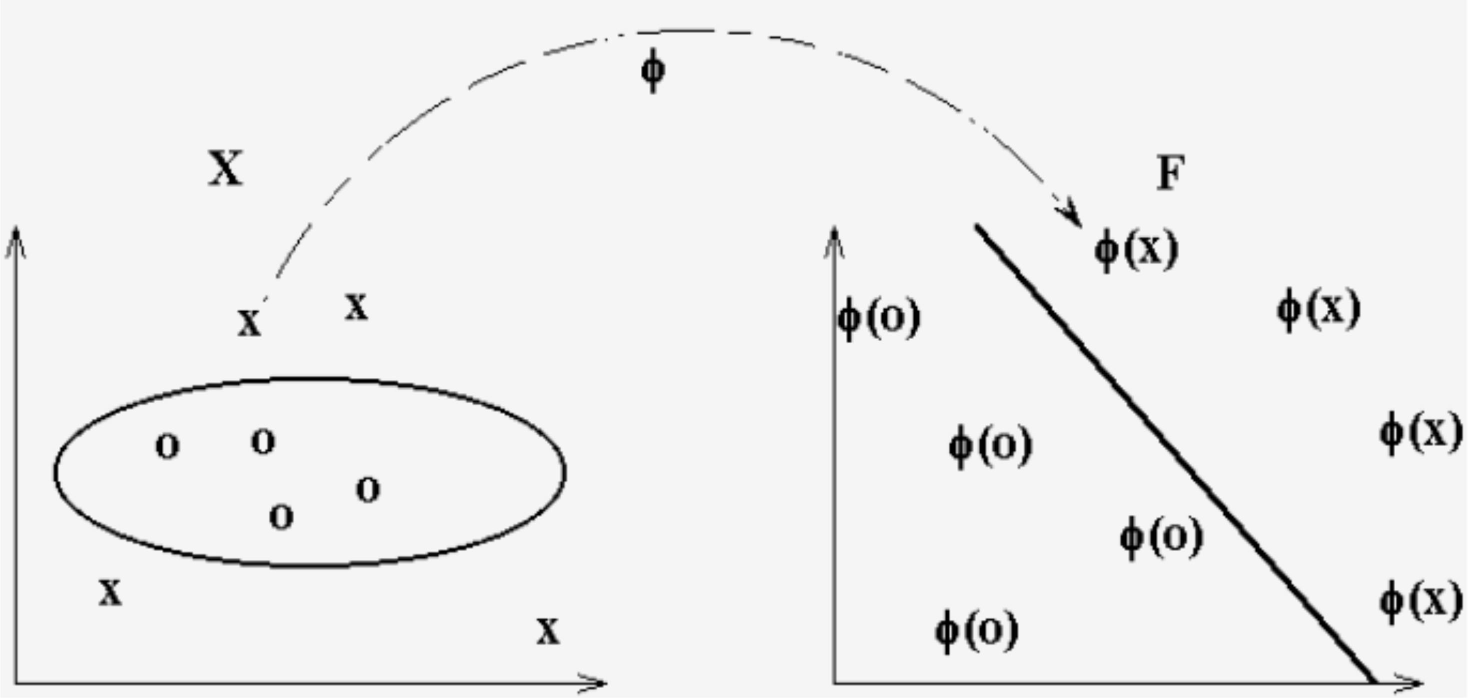
Non-Linear Classifier

- The support vector classifier is fairly easy to think about. However, because it only allows for a linear decision boundary it may not be all that powerful.



Support Vector Machines

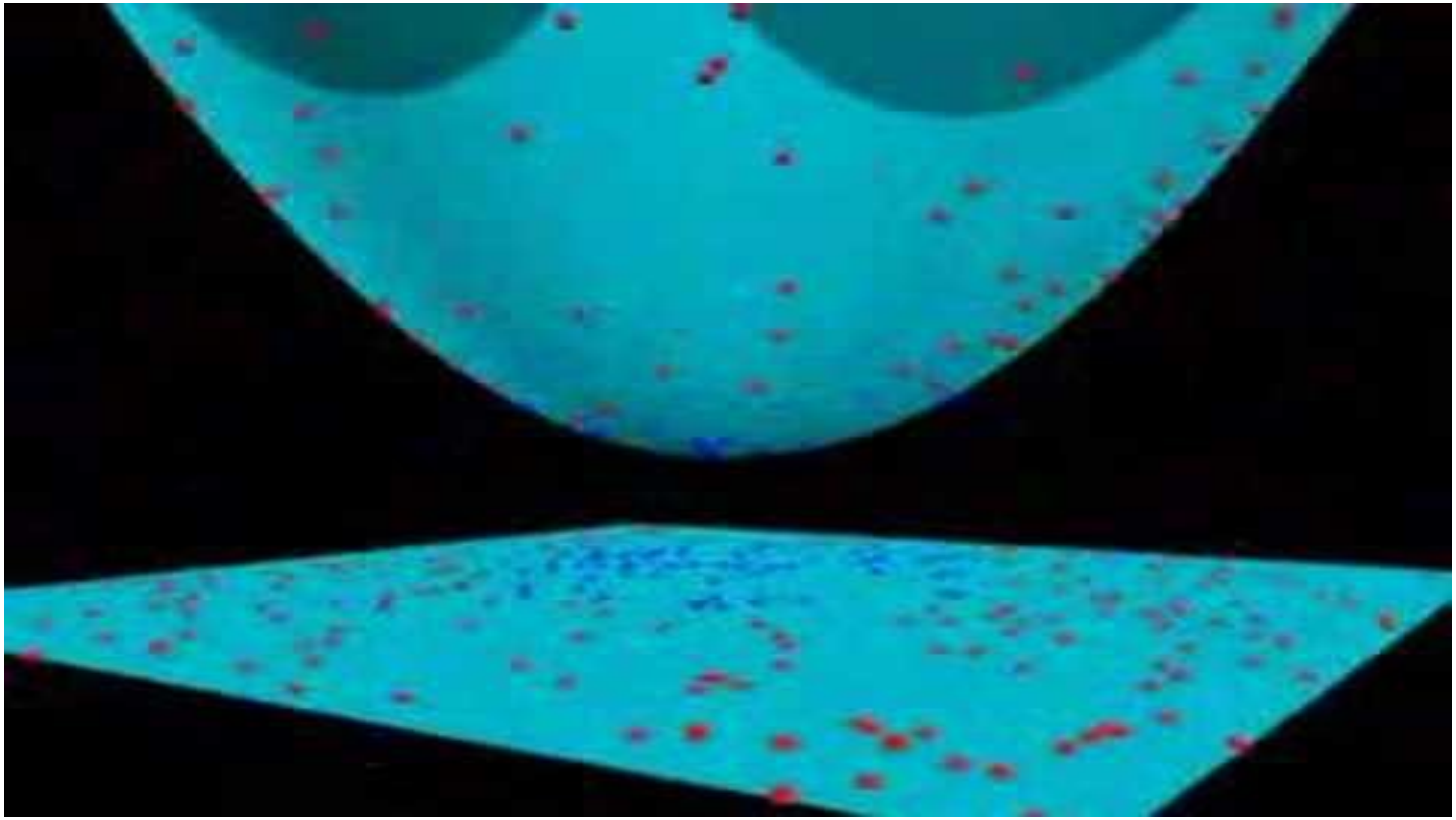
- SVM maps data into a high-dimensional feature space including non-linear features, then use a linear classifier there



In the original feature space:
Polynomial boundary

In the high-dimensional feature space:
Linear boundary

SVM Visualisation



https://www.youtube.com/watch?v=3liCbRZPrZA

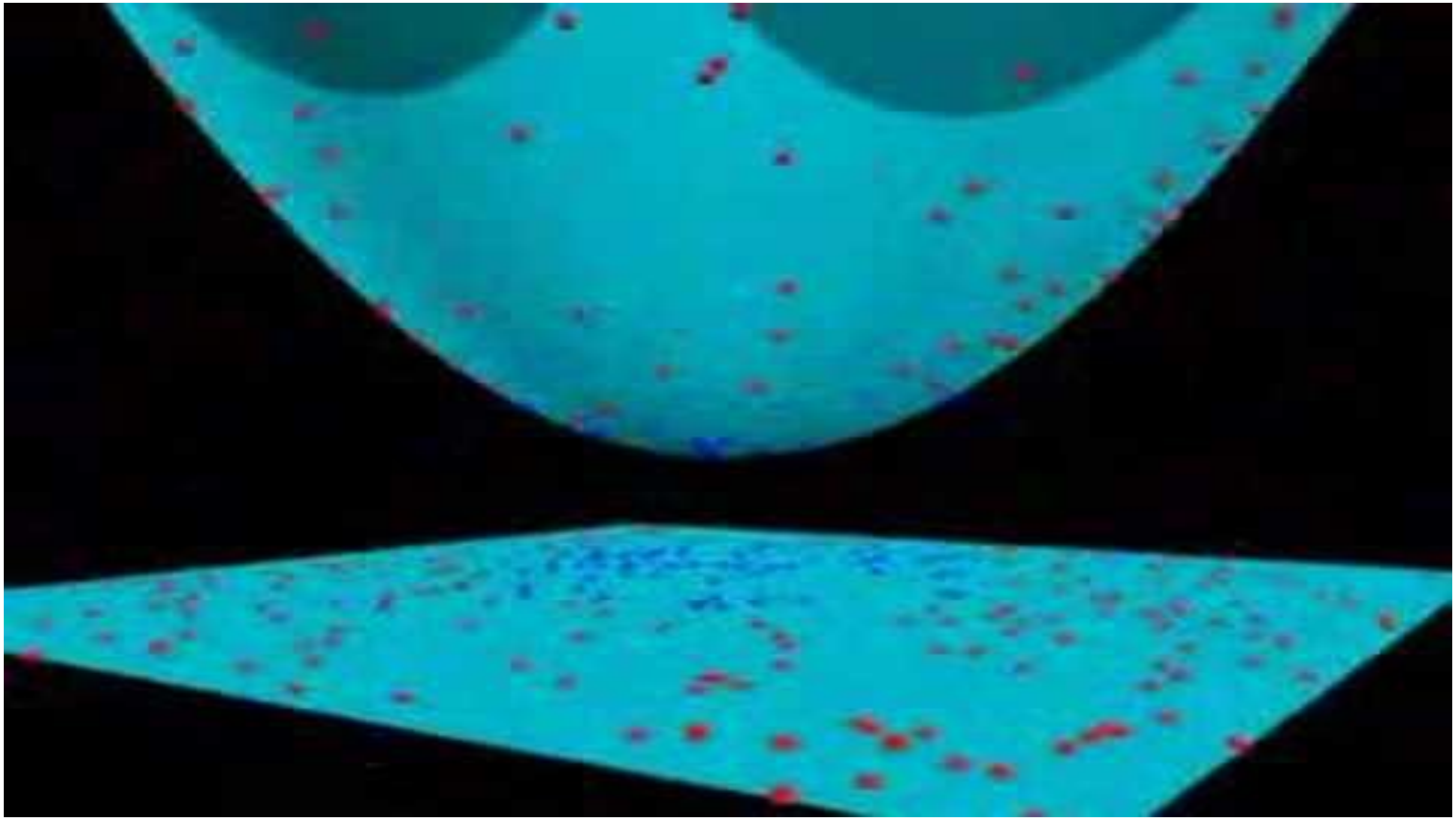
How SVM Works – An Example

- In the original feature space:
  - Two features: X_1, X_2
  - Quadratic function: $f(X_1,X_2) = 2X_1^2 - 3X_2^2 + X_1 + 5X_2 - 8$
- In the high-dimensional feature space:
  - Four features: Z_1, Z_2, Z_3, Z_4
  - Linear function: $f(Z_1, Z_2, Z_3, Z_4) = 2Z_1 - 3Z_2 + Z_3 + 5Z_4 - 8$
- Transformations
  - The function $f(Z_1, Z_2, Z_3, Z_4) = 2Z_1 - 3Z_2 + Z_3 + 5Z_4 - 8$ is
    - the optimal linear separating hyperplane obtained in the high-dimensional feature space
  - The **transformations (or a basis)** are as follows:
    - $Z_1=X_1^2$, $Z_2=X_2^2$, $Z_3=X_1$, $Z_4=X_2$
  - If we know the basis, then we can easily obtain
    - the optimal non-linear separating hyperplane in the original feature space
  - This is basically how SVM works.

In Reality

- While conceptually the basis approach is how the support vector machine works, there is some complicated math (which I will spare you) which means that we don’t actually choose the basis function.
- Instead we choose something called a Kernel function which takes the place of the basis.
- Common kernel functions include
  - Linear
  - Polynomial
  - Radial Basis Function
  - Sigmoid
- Pick a Kernel that represents your prior knowledge about the problem.

SVM with Polynomial Kernel Visualisation

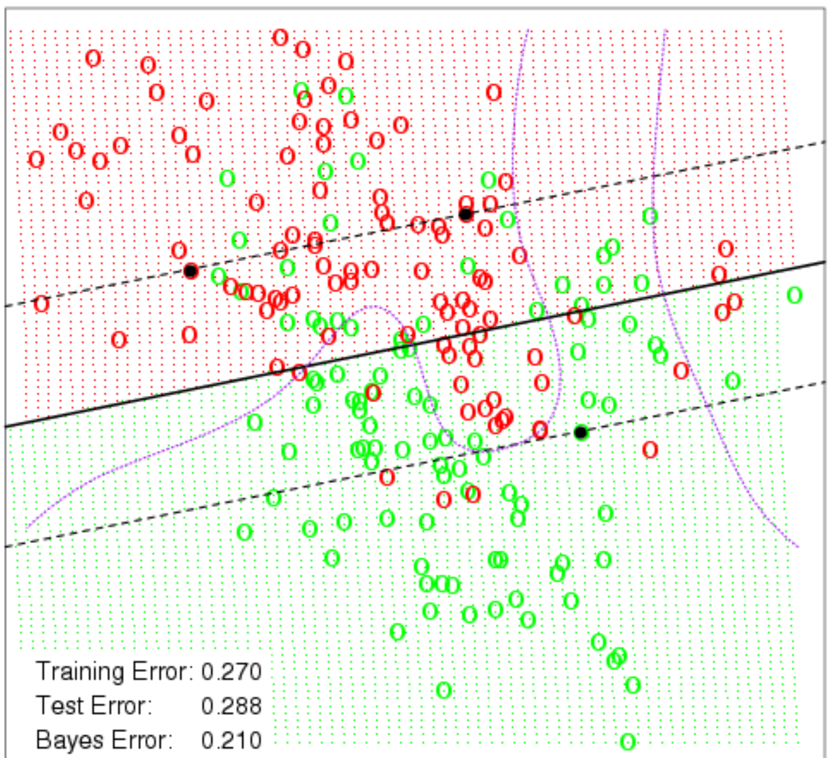


https://www.youtube.com/watch?v=3liCbRZPrZA

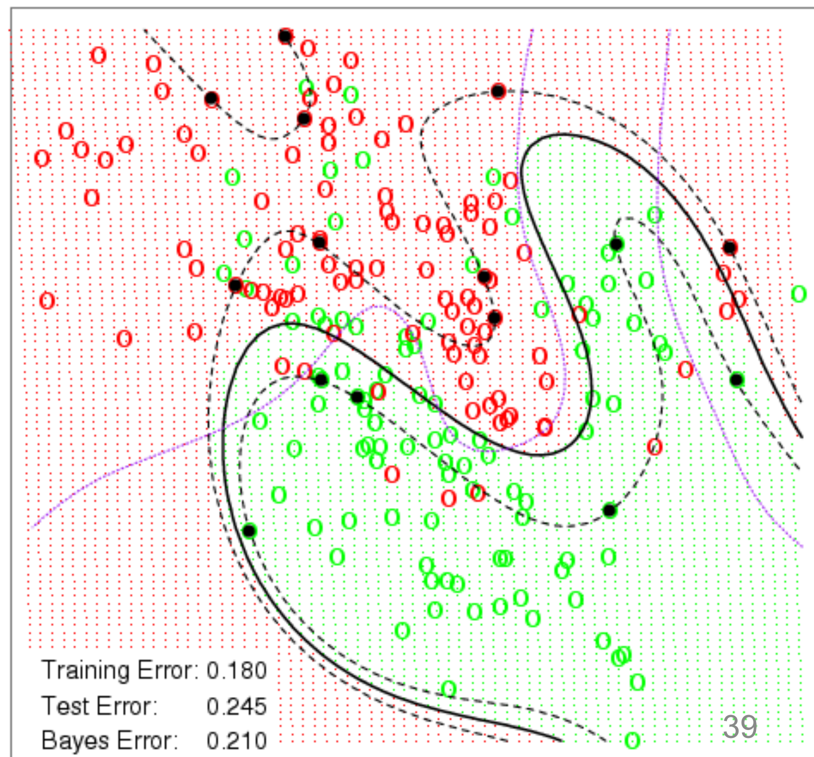
A Simulation Example

- This is the simulation example from Chapter 1.
- Using a polynomial kernel we now allow SVM to produce a non-linear decision boundary with a much lower test error rate.

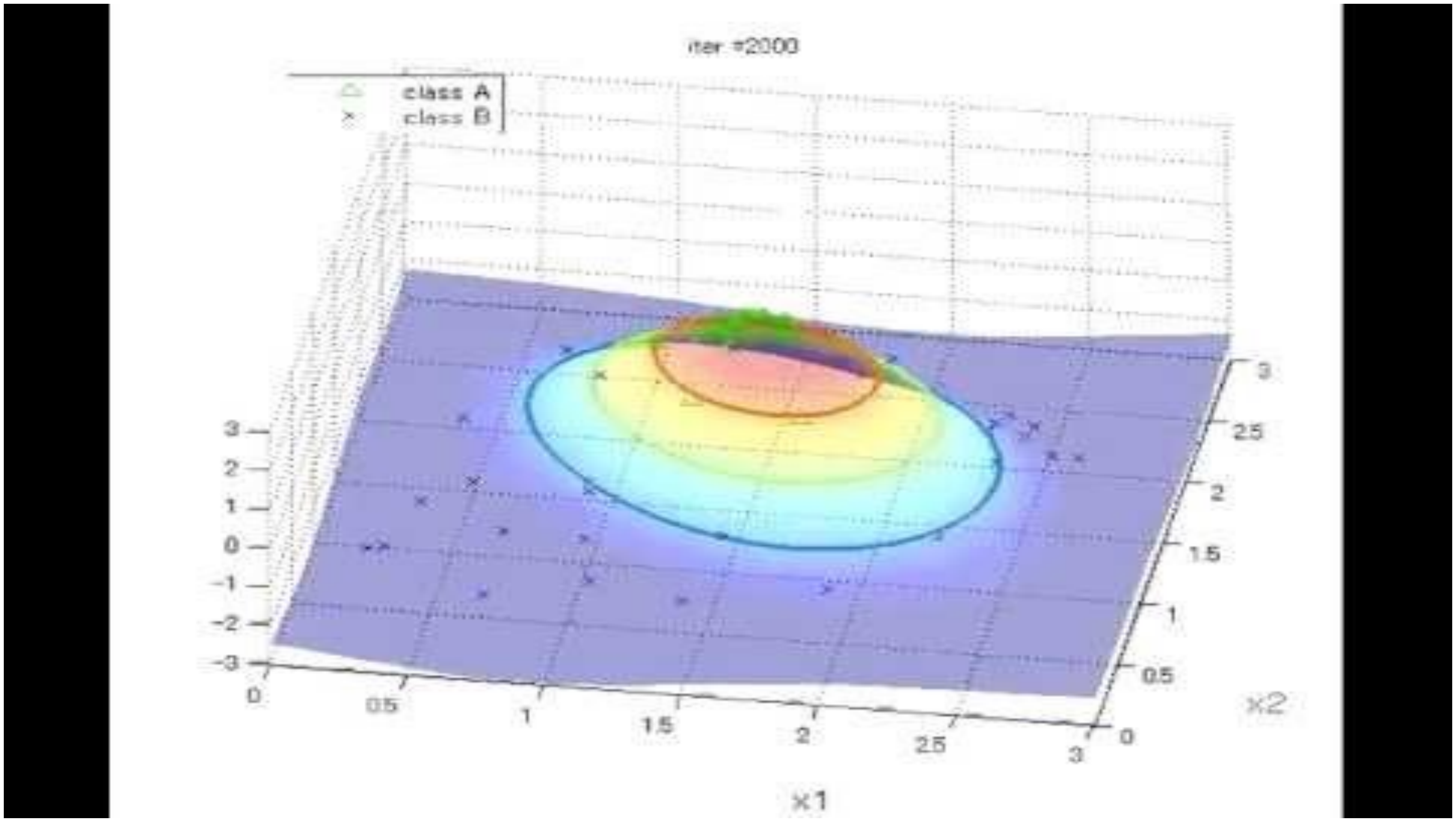
(The purple lines represent the Bayes decision boundaries)



SVM - Degree-4 Polynomial in Feature Space



SVM with Radial Kernel Visualisation

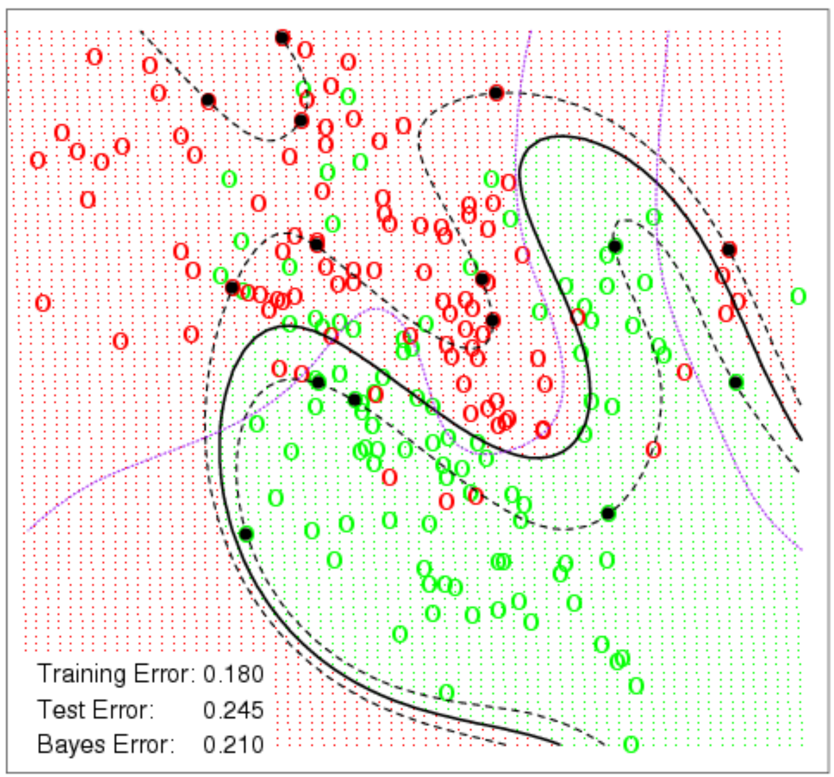


https://www.youtube.com/watch?v=NmhbQ-ag2z0

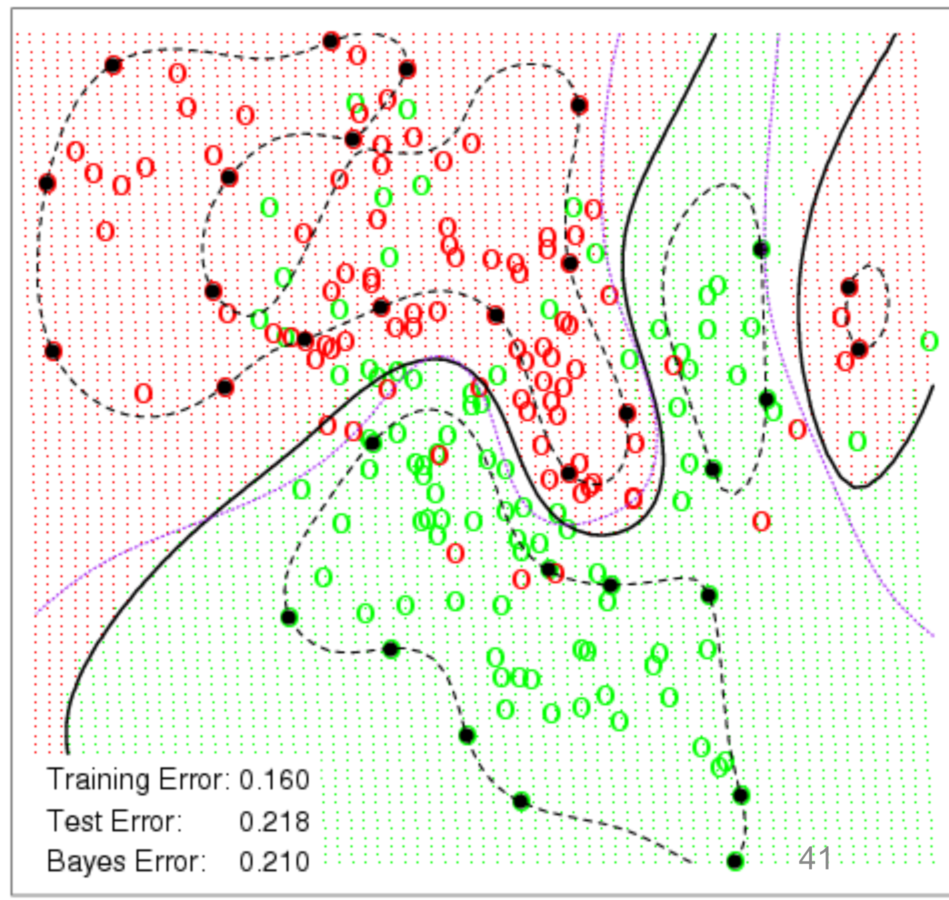
Radial Basis Kernel

- Using a Radial Basis Kernel you get an even lower error rate.

SVM - Degree-4 Polynomial in Feature Space



SVM - Radial Kernel in Feature Space

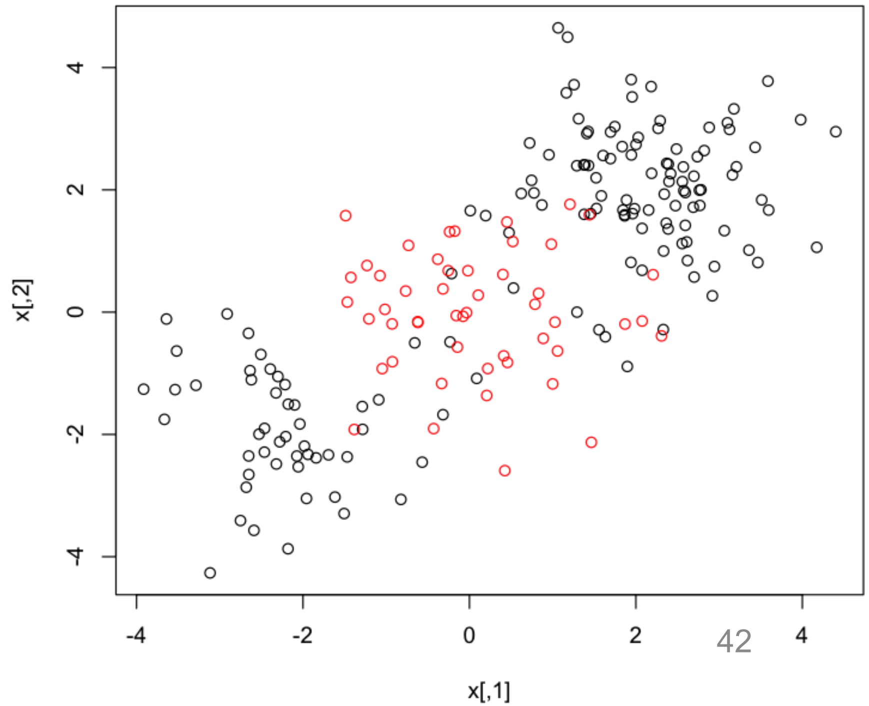


Support Vector Machine

- Change the value of `kernel` in the `svm()` function
  - Polynomial kernel: `kernel="polynomial"`
    - Use `degree` argument to specify a degree for the polynomial kernel
  - Radial kernel: `kernel="radial"`
    - Use `gamma` argument to specify a value of γ for the radial basis kernel

- First generate some data with a non-linear class boundary

```
> 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))
```



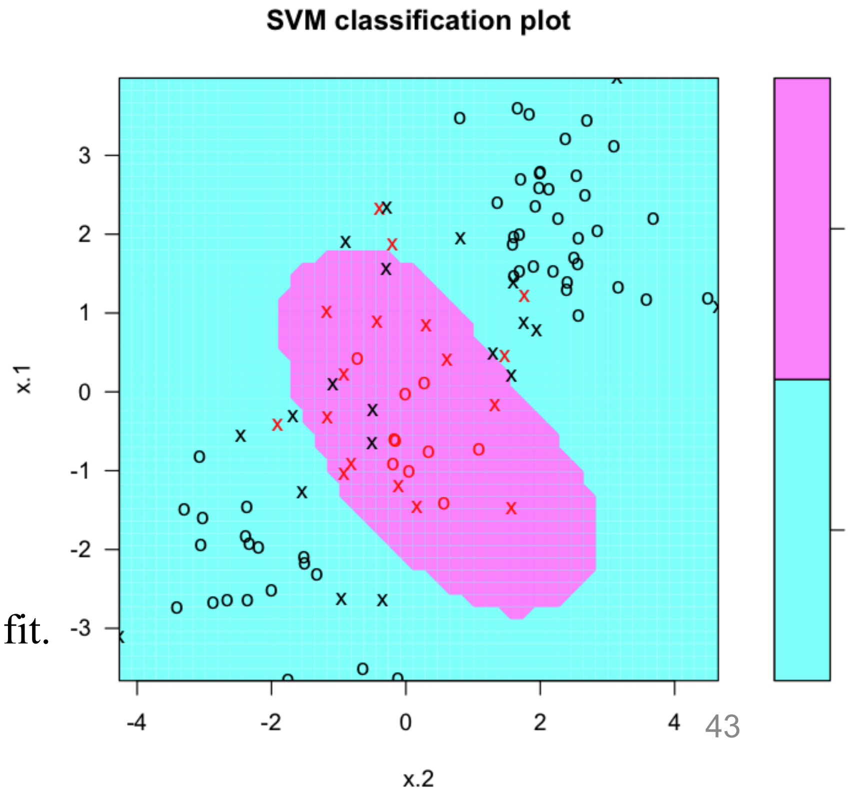
Support Vector Machine Example

```
> train=sample(200,100)
> svmfit=svm(y~.,data=dat[train,],kernel="radial",gamma=1,cost=1)
> plot(svmfit,dat[train,])
> summary(svmfit)
```

#randomly split into training and testing groups

#gamma is the value of γ for the radial basis kernel

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

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

Number of Support Vectors:  37
 ( 17 20 )

Number of Classes:  2
Levels:
 1 2
```

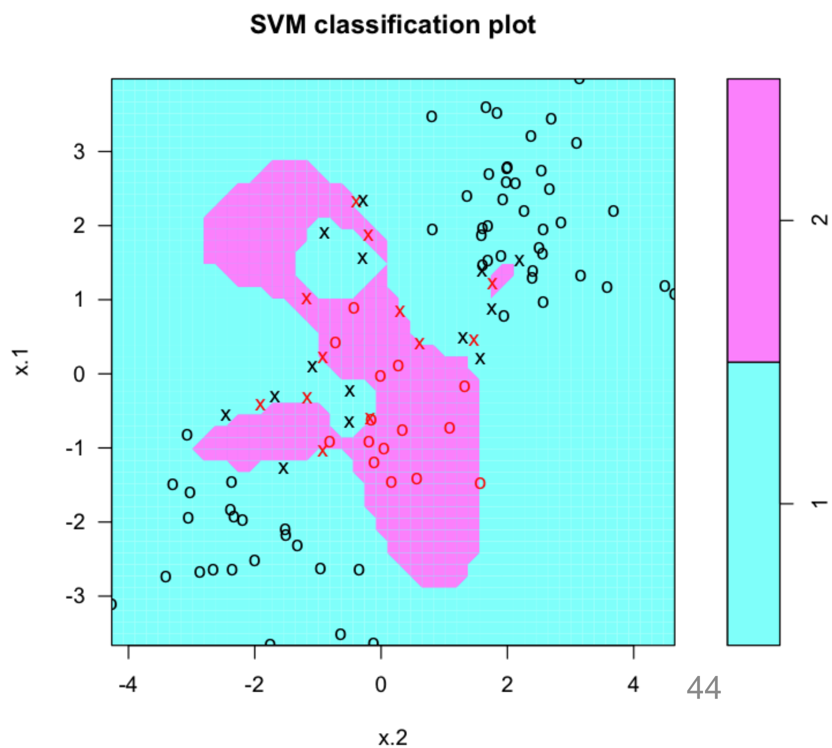


There are a fair number of training errors in this SVM fit.

Support Vector Machine Example

- What will happen if we increase the value of `cost`?
  - Reduce the number of training errors
  - More irregular boundary → risk of overfitting the data

```
> svmfit=svm(y~.,data=dat[train,],kernel="radial",gamma=1,cost=1e5)
> plot(svmfit,dat[train,])
> summary(svmfit)

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

Number of Support Vectors:  26 ( 12 14 )
Number of Classes:  2
Levels:
 1 2
```



Choosing Best Parameter Values

- Choose the best choice of γ and cost for an SVM with a radial kernel

```
> 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)

Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost gamma
    1     2
- best performance: 0.12
- Detailed performance results:
    cost gamma error dispersion
1  1e-01   0.5  0.27 0.11595018
2  1e+00   0.5  0.13 0.08232726
3  1e+01   0.5  0.15 0.07071068
……
```

Predicting Class Labels

- We can view the test set predictions for this model by applying the `predict()` function to the data

We take the subset of the data frame using `-train` as an index set.

```
> table(true=dat[-train,"y"],pred=predict(tune.out$best.model,newx=dat[-train,]))
     pred
true  1  2
   1 56 21
   2 18  5
```

`39% of test observations are misclassified by this SVM.`

LAB

- In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the `Auto` data set.

  (a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

  (b) Fit a support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

  (c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

  (d) Make some plots to back up your assertions in (b) and (c).

  Hint: In the lab, we used the plot() function for svm objects only in cases with p = 2. When p > 2, you can use the plot() function to create plots displaying pairs of variables at a time. Essentially, instead of typing > plot(svmfit , dat) where svmfit contains your fitted model and dat is a data frame containing your data, you can type > plot(svmfit , dat , x1~x4) in order to plot just the first and fourth variables. However, you must replace x1 and x4 with the correct variable names. To find out more, type ?plot.svm.

Solution

```
library(ISLR)
library(e1071)

##
# Part (a), create a categorical variable
#
mpg01=ifelse(Auto$mpg>median(Auto$mpg),1,0)  # so far mpg01 is numeric
mpg01=as.factor(mpg01)   #turn a numeric value to a categorical value
Auto1=data.frame(Auto,mpg01)  #incorporate mpg01 into the dataset

#remove name and mpg from the dataset
Auto1$name=NULL  #remove name as it is nominal, has nothing to do with the values
Auto1$mpg=NULL  #remove mpg as it has become mpg01
```

Solution

```
#
# Part (b) linear kernel:
#
tune.out=tune(svm, mpg01~.,
data=Auto1,kernel="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5
,10,100,1000)))

#Some plots to explore with:
#
plot(tune.out$best.model, Auto1, weight ~ horsepower )
plot( tune.out$best.model, Auto1, cylinders ~ year )
plot( tune.out$best.model, Auto1, cylinders ~ horsepower )
```

Solution

```
#
# Part (c) radial kernel:
#
tune.out = tune( svm, mpg01 ~ ., data=Auto1, kernel="radial", ranges =
list( cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000),gamma=c(0.5,1,2,3,4) ) )
summary(tune.out)  # <- use this output to select the optimal cost value

plot( tune.out$best.model, Auto1, weight ~ horsepower )
plot( tune.out$best.model, Auto1, cylinders ~ year )

# Part (c) polynomial kernel:
#
tune.out = tune( svm, mpg01 ~ ., data=Auto1, kernel="polynomial", ranges =
list( cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000),degree=c(1,2,3,4,5) ) )
summary(tune.out)  # <- use this output to select the optimal cost value
```