NBA 4920/6921 Lecture 22

Support Vector Machines 2

Murat Unal

11/11/2021

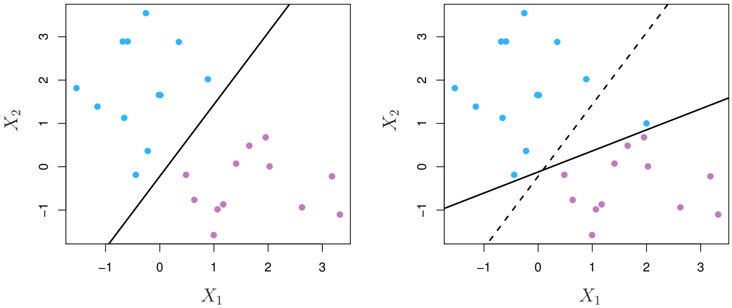
```r
rm(list=ls())
options(digits = 3, scipen = 999)
library(dplyr)
library(tidyverse)
library(ggplot2)
library(ISLR)
library(lmtest)
library(sandwich)
library(jtools)
library(caret)
library(ROCR)
library(e1071)
library(GGally)
set.seed(1)
```

Support Vector Machines

- Are a general class of classifiers that essentially attempt to separate two classes of observations
- The support vector machine generalizes a much simpler classifier—the maximal margin classifier
- The maximal margin classifier attempts to separate the two classes in our prediction space using a single hyperplane.

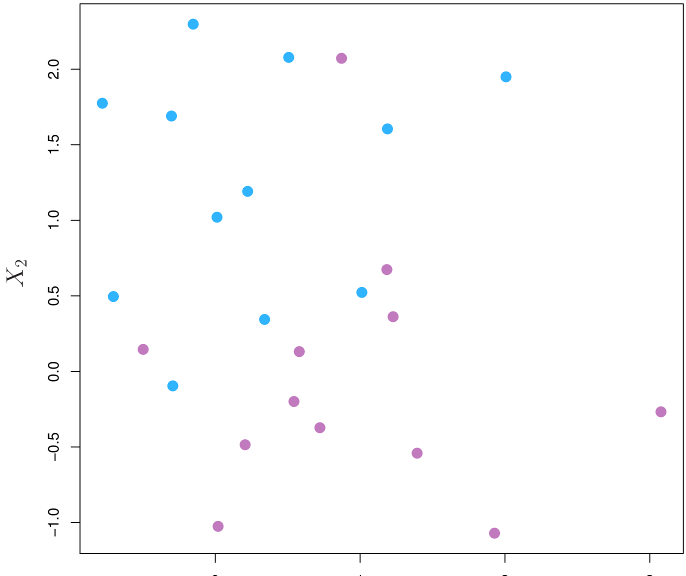
The maximal margin classifier

- The maximal margin hyperplane produces the maximal margin classifier
- The decision boundary only uses the support vectors—very sensitive



- This classifier can struggle in large dimensions

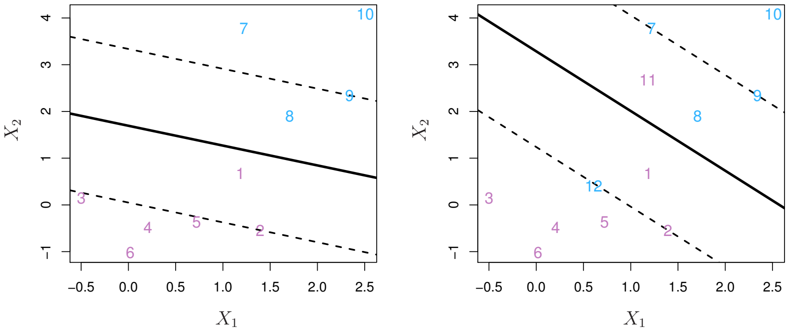
- In many cases no separating hyperplane exists, and so there is no maximal margin classifier



- However, we can extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called **soft margin**.

- The margin is **soft** because it can be violated by some of the training observations.
- The generalization of the maximal margin classifier to the non-separable case is known as the **support vector classifier**

Support Vector Classifier



- The hyperplane is shown as a solid line and the margins are shown as dashed lines
- Observations 3, 4, 5, and 6 are on the correct side of the margin, 2 is on the margin, and 1 is on the wrong side of the margin
- Observations 7 and 10 are on the correct side of the margin, 9 is on the margin, and 8 is on the wrong side of the margin.

- The support vector classifier classifies a test observation depending on which side of a hyperplane it lies.
- The hyperplane is chosen to correctly separate most of the training observations into the two classes, but may misclassify a few observations

- The support vector classifier selects a hyperplane by solving the problem
- Maximize the margin M over the set of $\beta_0, \beta_1, \cdots, \beta_p, \epsilon_1, \cdots, \epsilon_n, M$ such that

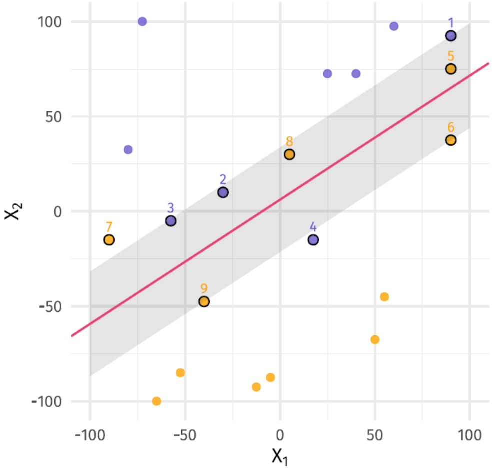
$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{1}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \tag{2}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \tag{3}$$

- M is the width of the margin; we seek to maximize this quantity
- ϵ_i are **slack variables** that allow i to violate the margin or hyperplane

- The slack variable ϵ_i tells us where the ith observation is located, relative to the hyperplane and relative to the margin
- If $\epsilon_i = 0$ then the ith observation is on the correct side of the margin
- If $\epsilon_i > 0$ then the ith observation is on the wrong side of the margin, and we say that the ith observation has violated the margin.
- If $\epsilon_i > 1$ then the ith observation is on the wrong side of the hyperplane



Support vectors

- On margin
- Violate margin
- Wrong side of hyperplane

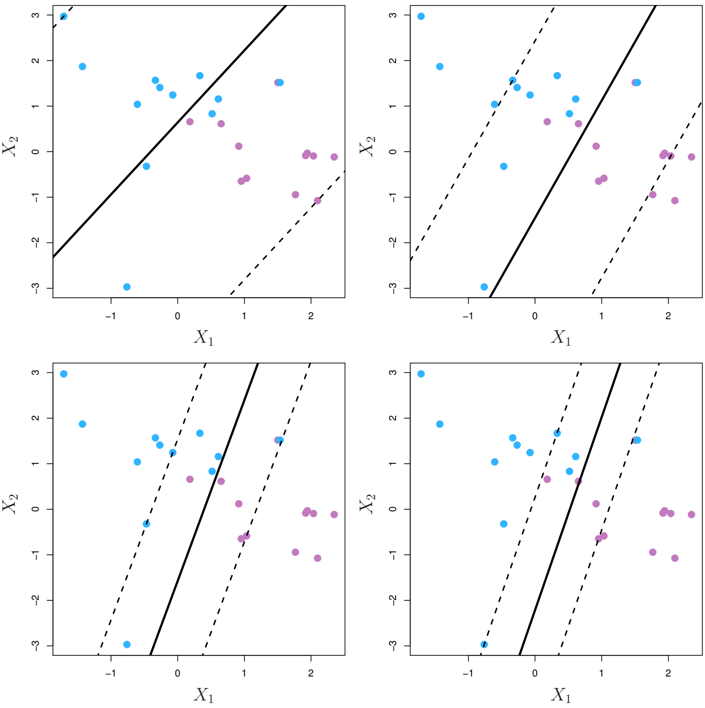
determine the classifier.

- C bounds the sum of the ϵ_i's, and so it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate.

- We can think of C as a budget for the amount that the margin can be violated by the n observations.

- If $C = 0$ then there is no budget for violations to the margin, and it must be the case that $\epsilon_1 = \cdots = \epsilon_n = 0$, in which case we're back to the maximal margin hyperplane optimization problem

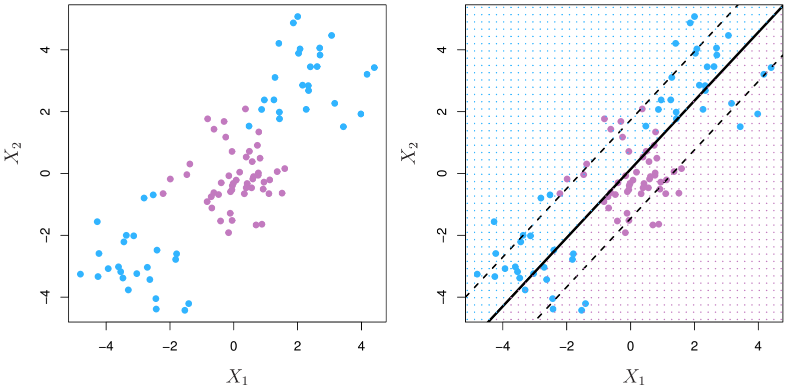
- As the budget C increases, we become more tolerant of violations to the margin, and so the margin will widen.
- Conversely, as C decreases, we become less tolerant of violations to the margin and so the margin narrows.
- In practice, C is treated as a tuning parameter that is generally chosen via cross-validation.



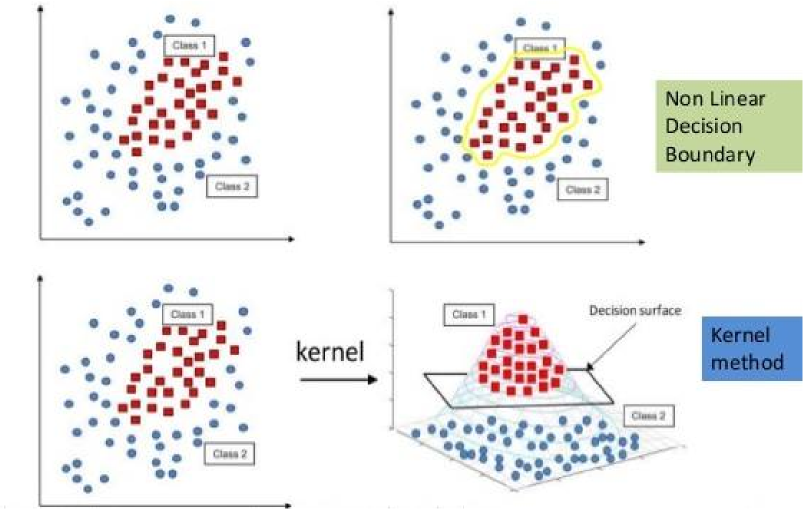
X_1
X_2
−1
0
1
2
−3
−2
−1
0
1
2
3

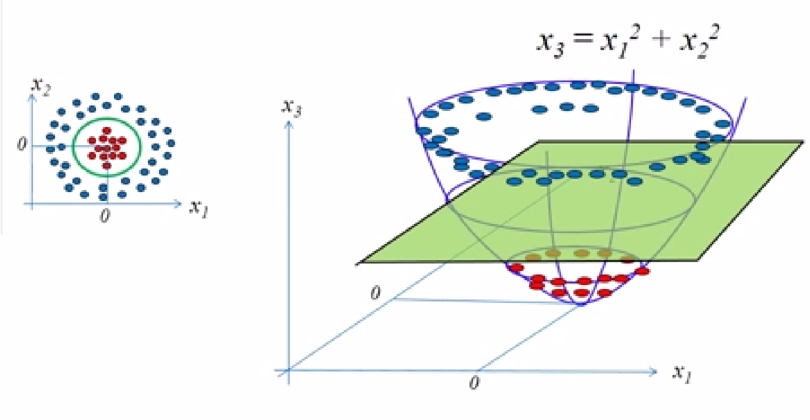
The Support Vector Machine

- The support vector classifier is a natural approach for classification in the two-class setting, if the boundary between the two classes is linear
- In practice we are often faced with non-linear class boundaries



- In the regression setting, we increase our model's flexiblity by adding polynomials in our predictors
- We can apply a very similar idea to the support vector classifier
- The new classifier has a linear decision boundary in the expanded space.
- The boundary is going to be nonlinear within the original space


Class 1
Class 2
Class 1
Class 2
Non Linear Decision Boundary
Class 1
Class 2
kernel
Class 1
Decision surface
Class 2
Kernel method



x_2
0
0
x_1
$x_3 = x_1^2 + x_2^2$
x_3
0
0
x_1

- The support vector machine (SVM) is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using **kernels**.
- The main idea is that we may want to enlarge our feature space in order to accommodate a non-linear boundary between the classes.
- The kernel approach is an efficient computational approach for enacting this idea.

Dot products

- The solution to the support vector classifier only involves the dot product of the observations.
- The dot product of two vectors is defined as

$$a \cdot b = \sum_{i=1}^{p} a_i b_i = a_1 b_1 + a_1 b_1 + \cdots + a_p b_p$$

- Dot product is a measure of similarity between two vectors

- The linear support vector classifier can be written as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i = x \cdot x_i$$

- We fit the n α_i and β_0 with the training observations' dot products.
- It turns out that $\alpha_i \neq 0$ only for support-vector observations.

- The linear support vector classifier can be written as

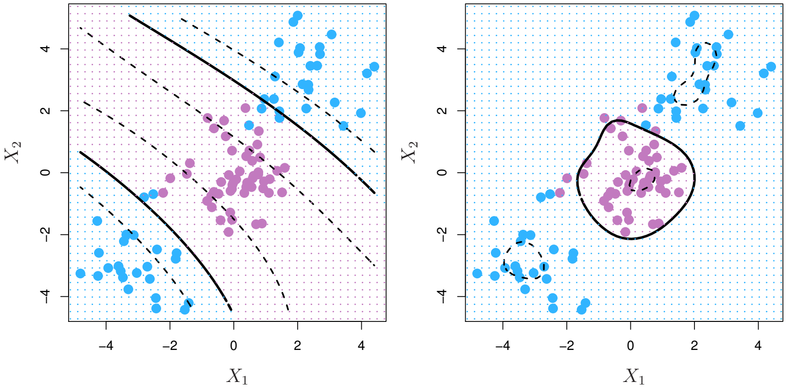
$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i = x \cdot x_i$$

- Support vector machines generalize this linear classifier by simply replacing $x \cdot x_i$ with **kernel functions**, $K(x_i, x_i')$

- **Kernel functions** offer alternative ways to measure the similarity between observations

1. Linear kernel : $K(x_i, x_i') = \sum_{j=1}^{p} x_{ij} x_{i'j}$
2. Polynomial kernel : $K(x_i, x_i') = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^2$
3. Radial kernel :$K(x_i, x_i') = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$

- Left: An SVM with a polynomial kernel of degree 3 is applied to the non-linear data.
- Right: An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.

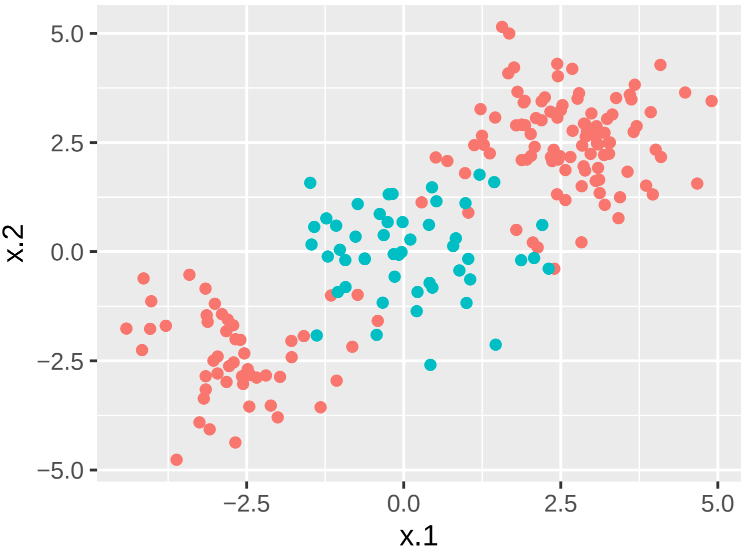


- Why use **kernel functions** if we instead could simply enlarge the feature space using functions on the original features?
- Computational advantage.
- For some kernels, such as the radial kernel, the feature space is implicit and infinite-dimensional, so we could never do the computations there anyway!

Application

```
x=matrix (rnorm (200*2) , ncol =2)
y=c(rep (1,150) , rep (2 ,50) )
x[1:100,]= x[1:100,] + 2.5
x[101:150,]= x[101:150,] - 2.5
data=data.frame(x=x,y=as.factor(y))
train = sample(200,100)
data_train = data[train,]
data_test = data[-train,]
```

- The two classes are not linearly separable

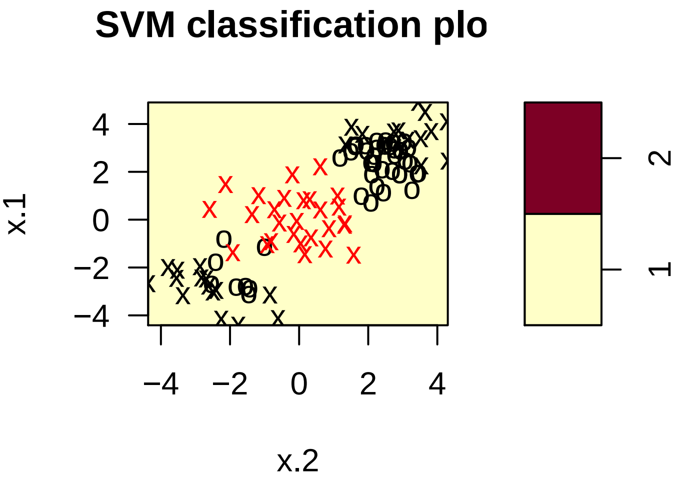


Linear kernel

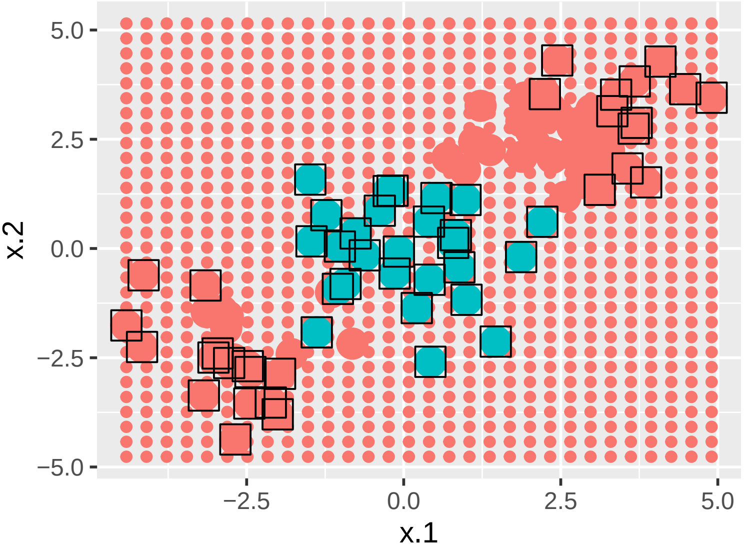
```r
svm.linear = tune(svm,y~ ., data=data_train , kernel ="line
             ranges =list(cost=10^seq(-3, 2, by = 0.5)))
summary(svm.linear)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
  cost
 0.001

- best performance: 0.27

- Detailed performance results:
        cost error dispersion
1    0.00100  0.27      0.134
2    0.00316  0.27      0.134
3    0.01000  0.27      0.134
```

```
plot(svmfit, data_train)
```



```r
make.grid = function(x, n = 30){
  grange = apply(x, 2, range)
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n
  expand.grid(x.1 = x1, x.2 = x2)}
xgrid = make.grid(x)
```



5.0
2.5
0.0
−2.5
−5.0
x.2
−2.5
0.0
2.5
5.0
x.1

- Make predictions on the test data

```r
ypred=predict(svmfit,data_test )
cm.linear <- confusionMatrix(data=ypred,
                reference=data_test$y,
                positive="2")
cm.linear$table
```

```
          Reference
Prediction  1  2
         1 77 23
         2  0  0
```

```r
cm.linear$overall[1]
```

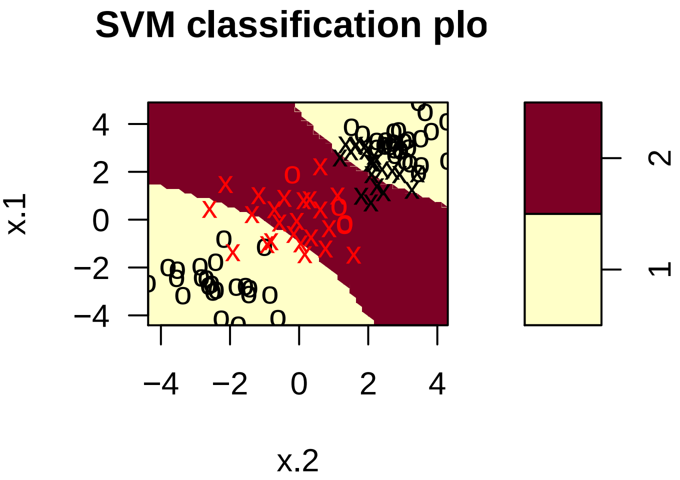
```
Accuracy 
    0.77 
```

Polynomial kernel

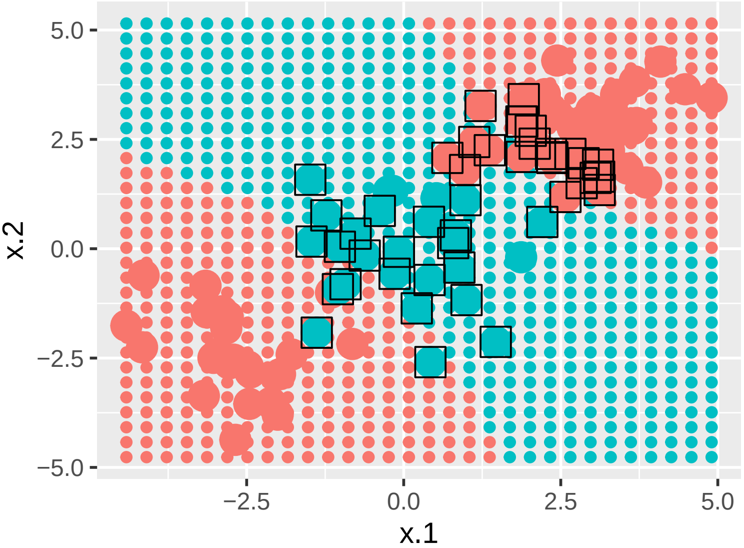
```
svm.poly = tune(svm,y~ ., data=data_train , kernel ="polyno
              ranges =list(cost=10^seq(-3, 2, by = 0.5),
                           degree=c(1,2,3,4,5)))
summary(svm.poly)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
    1      2

- best performance: 0.1

- Detailed performance results:
        cost degree error dispersion
1    0.00100      1  0.27     0.1418
2    0.00316      1  0.27     0.1418
```

```
plot(svmfit,data_train)
```



```r
make.grid = function(x, n = 30){
  grange = apply(x, 2, range)
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n
  expand.grid(x.1 = x1, x.2 = x2)}
xgrid = make.grid(x)
```


5.0
2.5
0.0
−2.5
−5.0
x.2
−2.5
0.0
2.5
5.0
x.1

- Make predictions on the test data

```
ypred=predict(svmfit,data_test )
cm.poly <- confusionMatrix(data=ypred,
                reference=data_test$y,
                positive="2")
cm.poly$table
```

```
          Reference
Prediction  1  2
         1 64  4
         2 13 19
```

```
cm.poly$overall[1]
```

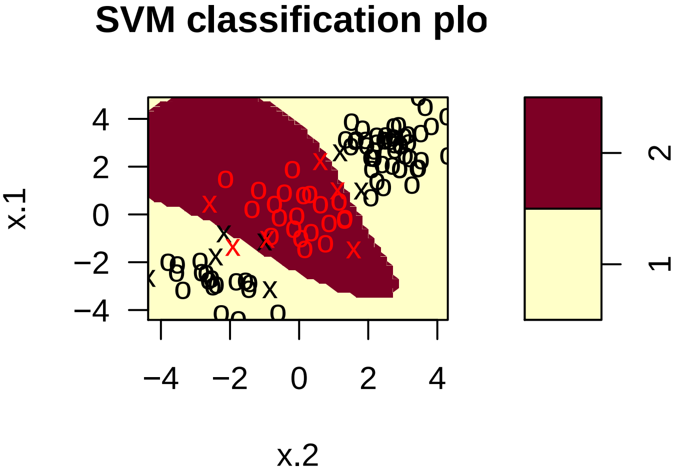
```
Accuracy
    0.83
```

Radial kernel

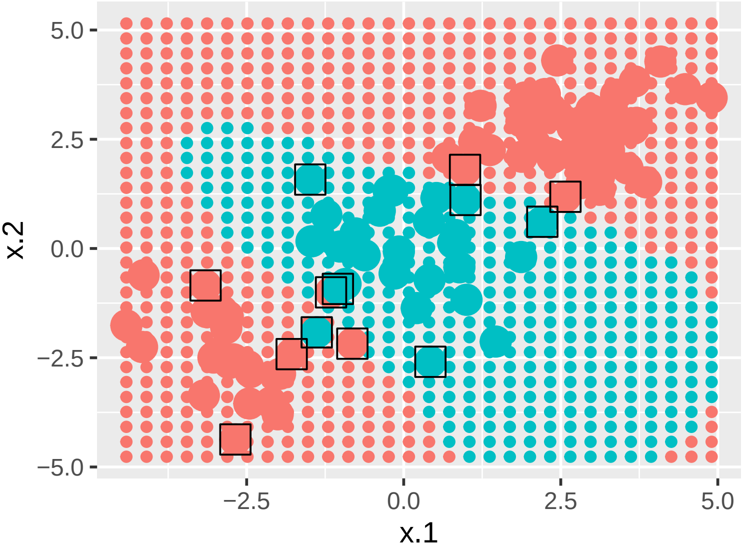
```
svm.rad = tune(svm,y~ ., data=data_train , kernel ="radial'
               ranges =list(cost=10^seq(-3, 2, by = 0.5),
                            gamma=c(0.5,1,2,3,4)))
summary(svm.rad)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
 31.6   0.5

- best performance: 0.03

- Detailed performance results:
        cost gamma error dispersion
1    0.00100   0.5  0.27     0.1567
2    0.00316   0.5  0.27     0.1567
```

```
plot(svmfit,data_train)
```



```
make.grid = function(x, n = 30){
  grange = apply(x, 2, range)
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n
  expand.grid(x.1 = x1, x.2 = x2)}
xgrid = make.grid(x)
```



5.0
2.5
0.0
−2.5
−5.0
x.2
−2.5
0.0
2.5
5.0
x.1

- Make predictions on the test data

```
ypred=predict(svmfit,data_test )
cm.radial <- confusionMatrix(data=ypred,
                    reference=data_test$y,
                    positive="2")
cm.radial$table
```

```
          Reference
Prediction  1  2
         1 68  2
         2  9 21
```

```
cm.radial$overall[1]
```

```
Accuracy
    0.89
```

- Compare performances

```
c(cm.linear$overall[1],cm.linear$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
       0.77        0.00        1.00          NA
```

```
c(cm.poly$overall[1],cm.poly$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.830       0.826       0.831       0.691
```

```
c(cm.radial$overall[1],cm.radial$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.890       0.913       0.883       0.792
```

Exercise

```
cars_train <- read.csv("cayugacars_train.csv")
cars_test <- read.csv("cayugacars_test.csv")
data_train <- cars_train[,-c(10:ncol(cars_train))]
data_test <- cars_test[,-c(10:ncol(cars_test))]
```

Linear kernel

```
svm.linear = tune(svm,customer_bid~ ., data=data_train , ke
summary(svm.linear)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
    cost
 0.00316

- best performance: 0.234

- Detailed performance results:
        cost error dispersion
1    0.00100 0.380     0.0646
2    0.00316 0.234     0.0374
3    0.01000 0.237     0.0431
4    0.03162 0.238     0.0446
```

```r
svmfit <- svm.linear$best.model
svmfit
```

```
Call:
best.tune(method = svm, train.x = customer_bid ~ ., data =
    ranges = list(cost = 10^seq(-3, 2, by = 0.5)), kernel =


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.00316

Number of Support Vectors:  878
```

- Make predictions on the test data

```r
ypred=predict(svmfit,data_test )
cm.linear <- confusionMatrix(data=ypred,
                reference=data_test$customer_bid,
                positive="Yes")
cm.linear$table
```

```
          Reference
Prediction  No Yes
       No  168  45
       Yes  20  74
```

```r
cm.linear$byClass[7]
```

```
   F1
0.695
```

Polynomial kernel

```
svm.poly = tune(svm,customer_bid~ ., data=data_train ,
              ranges =list(cost=10^seq(-3, 2, by = 0.5),
                           degree=c(2,3,4)))
summary(svm.poly)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
    1      3

- best performance: 0.247

- Detailed performance results:
        cost degree error dispersion
1    0.00100      2 0.386     0.0326
2    0.00316      2 0.386     0.0326
```

```r
svmfit <- svm.poly$best.model
svmfit
```

```
Call:
best.tune(method = svm, train.x = customer_bid ~ ., data =
    ranges = list(cost = 10^seq(-3, 2, by = 0.5), degree =
        4)), kernel = "polynomial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
     coef.0:  0

Number of Support Vectors:  703
```

- Make predictions on the test data

```
ypred=predict(svmfit,data_test )
cm.poly <- confusionMatrix(data=ypred,
                    reference=data_test$customer_bid,
                    positive="Yes")
cm.poly$table
```

```
          Reference
Prediction  No Yes
       No  156  40
       Yes  32  79
```

```
cm.poly$byClass[7]
```

```
   F1
0.687
```

Radial kernel

```
svm.rad = tune(svm,customer_bid~ ., data=data_train,
            kernel ="radial",
            ranges =list(cost=10^seq(-3, 2, by = 0.5),
                         gamma=c(0.01,0.05,0.1,0.5,1,2)))
summary(svm.rad)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
  cost gamma
 0.316   0.1

- best performance: 0.221

- Detailed performance results:
        cost gamma error dispersion
1    0.00100  0.01 0.386     0.0473
```

```r
svmfit <- svm.rad$best.model
svmfit
```

```
Call:
best.tune(method = svm, train.x = customer_bid ~ ., data = 
    ranges = list(cost = 10^seq(-3, 2, by = 0.5), gamma = c
        0.05, 0.1, 0.5, 1, 2)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.316

Number of Support Vectors:  723
```

- Make predictions on the test data

```
ypred=predict(svmfit,data_test )
cm.radial <- confusionMatrix(data=ypred,
                reference=data_test$customer_bid,
                positive="Yes")
cm.radial$table
```

```
          Reference
Prediction  No Yes
       No  168  43
       Yes  20  76
```

```
cm.radial$byClass[7]
```

```
   F1
0.707
```

- Compare performances

```
c(cm.linear$overall[1],cm.linear$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.788       0.622       0.894       0.695
```

```
c(cm.poly$overall[1],cm.poly$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.765       0.664       0.830       0.687
```

```
c(cm.radial$overall[1],cm.radial$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.795       0.639       0.894       0.707
```