# R programming: first steps

### Preparing the environment

rm(list = ls()); # Removing old variables in the workspace

library(caret); # for machine learning

library(e1071); # to use svm

### Loading data from package

data(iris); #to load dataset

### Printing the dataset

# print( iris ); # to print the data table

print( head(iris) ); # to print just first rows of the data table

print( iris[5:15, ] ); # to print just selected rows of the table

### Loading data from csv file

D <- read.csv("C:/Users/henrytu/Desktop/example1.csv", header=TRUE);

print(D);

### Fitting / learning the linear model

M <- lm(y ~ x, D);

print(M);

### Testing the model if it fits the dataset

y <- D$y; #The ground-truth values in the dataset

y1 <- predict(M, D); #The predict valuese by the model M

e1 <- abs(y - y1);

print( data.frame(y, y1, e1));

print( sum(e1) );

### We will have a lot of error

|  |  |
| --- | --- |
| y y1 e1  1 1.0 -13.859091 14.8590909  2 4.0 -1.869091 5.8690909  3 9.3 10.120909 0.8209091  4 16.4 22.110909 5.7109091  5 25.0 34.100909 9.1009091  6 36.0 46.090909 10.0909091  7 49.0 58.080909 9.0809091  8 64.3 70.070909 5.7709091  9 81.0 82.060909 1.0609091  10 100.0 94.050909 5.9490909  11 121.0 106.040909 14.9590909 | How to fix this problem?  How to make the error smaller? |

### We add one more column (transformation)

D$xx <- D$x \* D$x;

print(D);

### And we fit the new model to the new data

M <- lm(y ~ x + xx, D); # or you can write M <- lm(y ~ ., D);

print(M);

### Comparing the last error and this time

|  |  |
| --- | --- |
| y y1 e1  1 1.0 1.062238 0.06223776  2 4.0 4.099441 0.09944056  3 9.3 9.126154 0.17384615  4 16.4 16.142378 0.25762238  5 25.0 25.148112 0.14811189  6 36.0 36.143357 0.14335664  7 49.0 49.128112 0.12811189  8 64.3 64.102378 0.19762238  9 81.0 81.066154 0.06615385  10 100.0 100.019441 0.01944056  11 121.0 120.962238 0.03776224  > print( sum(e1) );  [1] 1.333706 | y y1 e1  1 1.0 -13.859091 14.8590909  2 4.0 -1.869091 5.8690909  3 9.3 10.120909 0.8209091  4 16.4 22.110909 5.7109091  5 25.0 34.100909 9.1009091  6 36.0 46.090909 10.0909091  7 49.0 58.080909 9.0809091  8 64.3 70.070909 5.7709091  9 81.0 82.060909 1.0609091  10 100.0 94.050909 5.9490909  11 121.0 106.040909 14.9590909  > print( sum(e1) );  [1] 83.27273 |
| Call:  lm(formula = y ~ x + xx, data = D)  Coefficients:  (Intercept) x xx  0.01455 0.05294 0.99476 | Call:  lm(formula = y ~ x, data = D)  Coefficients:  (Intercept) x  -25.85 11.99 |

### Conclusion

+ The quadratic model y = 0.01455 + 0.05294x + 0.99476xx has smaller error subject to the data in the file example1.csv

# R programming: using function

rm(list = ls()); # to remove variable

library(caret); # for machine learning

library(e1071); # to use svm

### How to write functions in R for testing

|  |
| --- |
| testLinear <- function(D)  {  M <- lm(y ~ x, D);  print(M);  y <- D$y; #The ground-truth values in the dataset  y1 <- predict(M, D); #The predict valuese by the model M  e1 <- abs(y - y1);  print( data.frame(y, y1, e1));  print( sum(e1) );  }  testQuadric <- function(D)  {  D$xx <- D$x \* D$x;  print(D);  M <- lm(y ~ x + xx, D);  print(M);  y <- D$y; #The ground-truth values in the dataset  y1 <- predict(M, D); #The predict valuese by the model M  e1 <- abs(y - y1);  print( data.frame(y, y1, e1));  print( sum(e1) );  } |

### Using functions

|  |
| --- |
| D <- read.csv("C:/Users/henrytu/Desktop/example1.csv", header=TRUE);  testLinear(D);  testQuadric(D); |

# SVM for iris

### Loading data

rm(list=ls()); library(caret); library(e1071);

data(iris);

set.seed(4321);

### Train the model

A <- iris;

x <- subset(A, select=-c(Species));

y <- subset(A, select=c(Species));

M <- svm(x, y, type='C', kernel='polynomial', degree=3);

print(M);

### Test the model

x <- subset(A, select=-c(Species));

y <- A[, "Species"];

y1 <- predict (M, x);

e1 <- abs(y1 != y);

print(head(data.frame( y, y1, e1 )));

print(data.frame( key="Total error", val=sum(e1) ));

table(y1, y);

# SVM for iries: training and testing

### Loading the data

data(iris);

### Splitting the data into two parts

set.seed(4321);

S <- createDataPartition(iris$Species, p=0.7, list=FALSE);

A <- iris[S, ];

B <- iris[-S, ];

|  |  |  |
| --- | --- | --- |
| > set.seed(4321);  > S <- createDataPartition(iris$Species, p=0.7, list=FALSE);  > A <- iris[S, ]; B <- iris[-S, ];  >  > print(head(A)); print(nrow(A));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  2 4.9 3.0 1.4 0.2 setosa  3 4.7 3.2 1.3 0.2 setosa  4 4.6 3.1 1.5 0.2 setosa  5 5.0 3.6 1.4 0.2 setosa  8 5.0 3.4 1.5 0.2 setosa  10 4.9 3.1 1.5 0.1 setosa  [1] 105  > print(head(B)); print(nrow(B));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  1 5.1 3.5 1.4 0.2 setosa  6 5.4 3.9 1.7 0.4 setosa  7 4.6 3.4 1.4 0.3 setosa  9 4.4 2.9 1.4 0.2 setosa  11 5.4 3.7 1.5 0.2 setosa  14 4.3 3.0 1.1 0.1 setosa  [1] 45 | > set.seed(123);  > S <- createDataPartition(iris$Species, p=0.7, list=FALSE);  > A <- iris[S, ]; B <- iris[-S, ];  >  > print(head(A)); print(nrow(A));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  1 5.1 3.5 1.4 0.2 setosa  2 4.9 3.0 1.4 0.2 setosa  3 4.7 3.2 1.3 0.2 setosa  4 4.6 3.1 1.5 0.2 setosa  7 4.6 3.4 1.4 0.3 setosa  9 4.4 2.9 1.4 0.2 setosa  [1] 105  > print(head(B)); print(nrow(B));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  5 5.0 3.6 1.4 0.2 setosa  6 5.4 3.9 1.7 0.4 setosa  8 5.0 3.4 1.5 0.2 setosa  10 4.9 3.1 1.5 0.1 setosa  12 4.8 3.4 1.6 0.2 setosa  16 5.7 4.4 1.5 0.4 setosa  [1] 45 | > set.seed(125);  > S <- createDataPartition(iris$Species, p=0.7, list=FALSE);  > A <- iris[S, ]; B <- iris[-S, ];  >  > print(head(A)); print(nrow(A));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  1 5.1 3.5 1.4 0.2 setosa  2 4.9 3.0 1.4 0.2 setosa  3 4.7 3.2 1.3 0.2 setosa  4 4.6 3.1 1.5 0.2 setosa  6 5.4 3.9 1.7 0.4 setosa  8 5.0 3.4 1.5 0.2 setosa  [1] 105  > print(head(B)); print(nrow(B));  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  5 5.0 3.6 1.4 0.2 setosa  7 4.6 3.4 1.4 0.3 setosa  9 4.4 2.9 1.4 0.2 setosa  10 4.9 3.1 1.5 0.1 setosa  11 5.4 3.7 1.5 0.2 setosa  13 4.8 3.0 1.4 0.1 setosa  [1] 45 |

### Comparing different splits

|  |  |  |
| --- | --- | --- |
| > set.seed(4321);  > S <- createDataPartition(iris$Species, p=0.7, list=FALSE);  > A <- iris[S, ]; B <- iris[-S, ];  > traintest(A, B);  Call:  svm.default(x = x, y = y, type = "C", kernel = "polynomial", degree = 3)  Parameters:  SVM-Type: C-classification  SVM-Kernel: polynomial  cost: 1  degree: 3  gamma: 0.25  coef.0: 0  Number of Support Vectors: 54  Training error: 0.04666667  Testing error: 0.02222222  (2%) | > set.seed(123);  > S <- createDataPartition(iris$Species, p=0.7, list=FALSE);  > A <- iris[S, ]; B <- iris[-S, ];  > traintest(A, B);  Call:  svm.default(x = x, y = y, type = "C", kernel = "polynomial", degree = 3)  Parameters:  SVM-Type: C-classification  SVM-Kernel: polynomial  cost: 1  degree: 3  gamma: 0.25  coef.0: 0  Number of Support Vectors: 54  Training error: 0.04666667  Testing error: 0.04444444  (4%) | > set.seed(125);  > S <- createDataPartition(iris$Species, p=0.7, list=FALSE);  > A <- iris[S, ]; B <- iris[-S, ];  > traintest(A, B);  Call:  svm.default(x = x, y = y, type = "C", kernel = "polynomial", degree = 3)  Parameters:  SVM-Type: C-classification  SVM-Kernel: polynomial  cost: 1  degree: 3  gamma: 0.25  coef.0: 0  Number of Support Vectors: 54  Training error: 0.04666667  Testing error: 0.02222222 |

### Conclusion

+ In all the test, we have low training and low testing error.

+ So we have good SVM model ( type = "C", kernel = "polynomial", degree = 3 )