Support Vector Machines

Nik Bear Brown

In this lesson we'll learn the theory behind using Linear Discriminant Analysis (LDA) as a supervised classification technique. We'll then use LDA to classify the UCI wine dataset in R.

# Additional packages needed

To run the code you may need additional packages.

* If necessary install the followings packages.

install.packages("ggplot2");  
install.packages("e1071");  
install.packages("kernlab");

require(ggplot2)

## Loading required package: ggplot2

require(e1071)

## Loading required package: e1071

require(kernlab)

## Loading required package: kernlab

##   
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':  
##   
## alpha

# Data

We will be using the dataset 'svm\_regression.csv' which is randomly generated. [svm\_regression.csv](http://nikbearbrown.com/YouTube/MachineLearning/M08/svm_regression.csv). T

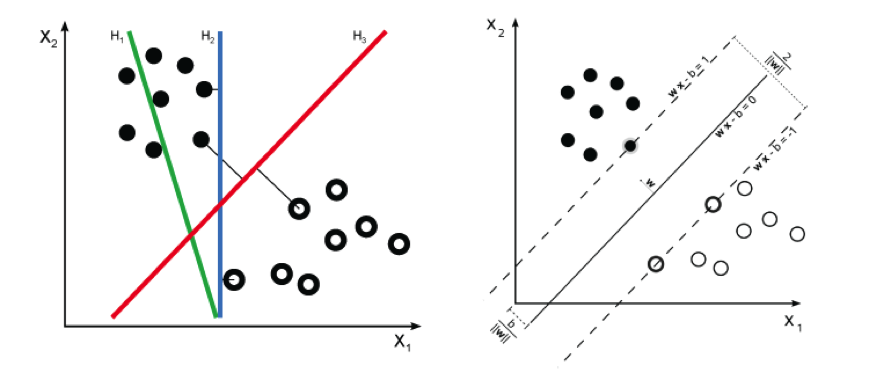
data\_url <- 'http://nikbearbrown.com/YouTube/MachineLearning/M08/svm\_regression.csv'  
svm.data <- read.csv(url(data\_url), sep=",", header = TRUE)  
head(svm.data)

## X Y  
## 1 1 3  
## 2 2 4  
## 3 3 8  
## 4 4 4  
## 5 5 6  
## 6 6 9

# Support Vector Machines

Support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Given a set of data points that belong to either of two classes, an SVM finds the hyperplane that:

* Leaves the largest possible fraction of points of the same class on the same side.
* Maximizes the distance of either class from the hyperplane.
* Find the optimal separating hyperplane that minimizes the risk of misclassifying the training samples and unseen test samples.

  
*SVM*

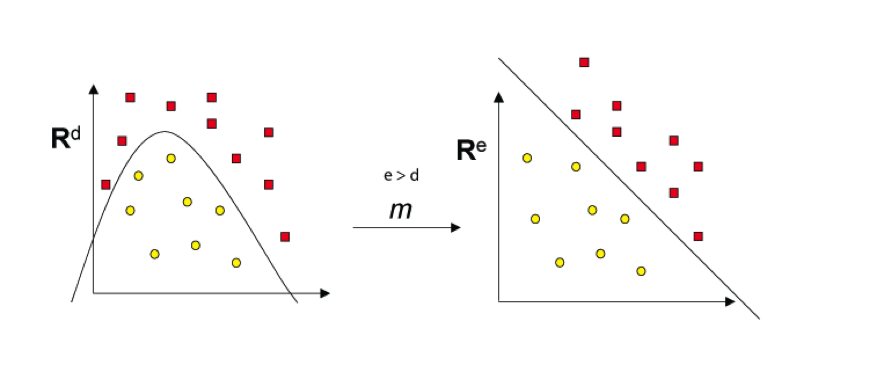
Pseudocode

Given a guess of width and bias we can:

* Compute whether all data points are in the correct half-planes.
* Compute the width of the margin.
* Search the space of width’s and bias to find the widest margin that matches all the datapoints.

## Kernels

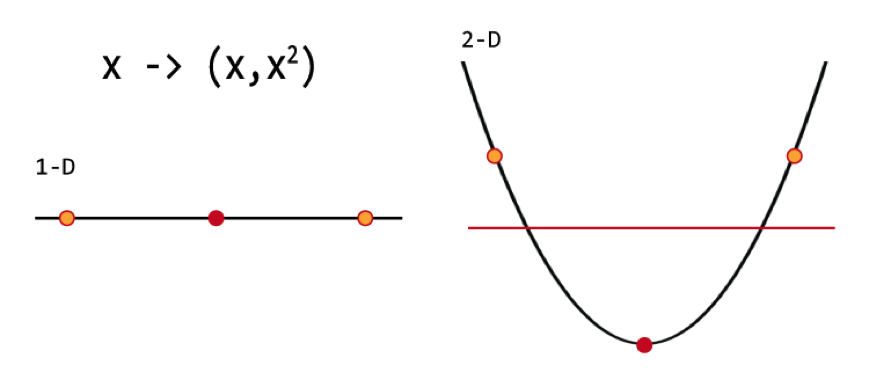
It is much easier and efficient to find Separating boundaries which are in the form of a straight lines as oppossed to curvy Separating boundaries. Kernels help us turn a linear classifier into a non-linear one. That is, they transform a curvy separating boundary that is only non-linearly separable to a linearly separable discriminant in a higher dimensional space.



Kernels

### Kernels (1-D Example)

For example, squaring the data may take some points that are not linearly separable in 1-D space to be linearly separable in 2D space.



Kernels

## Kernel Trick

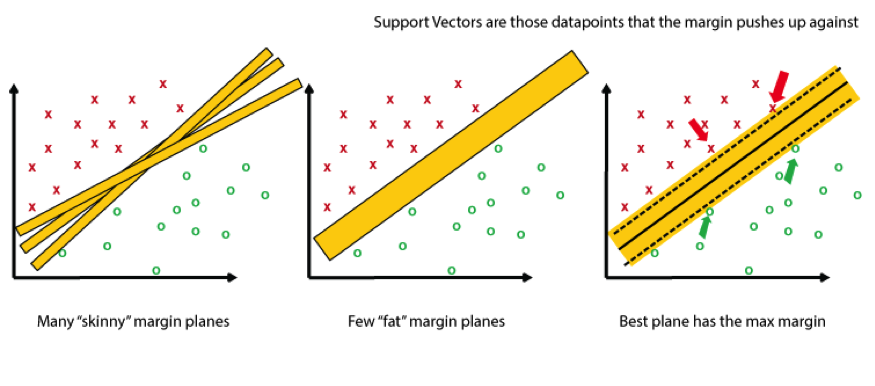
While coming up with functions that map linearly separable sets to SVM, we only need to know the inner product of vectors in the linearly separable sets in higher dimensional spaces arbitrarily coordinate space.

Computing the kernel of x and y gives the same result as the dot product in the mapped (high-dimensional) space. The mappings used by SVM schemes are designed to ensure that dot products may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function k(x,y) selected to suit the problem. This "Kernel trick" is" essentially is to define a similarity function in terms of original space itself without even defining (or even knowing), what the transformation function K will be.

## Linear SVM

Support Vectors are those datapoints that the margin pushes up against

Given some training data , a set of n points of the form where the is either 1 or −1, indicating the class to which the point belongs. Each is a p-dimensional real vector. We want to find the maximum-margin hyperplane that divides the points having from those having . Any hyperplane can be written as the set of points satisfying , where ,denotes the dot product of and , The variable is the (not necessarily normalized) normal vector to the hyperplane. The parameter determines the offset of the hyperplane from the origin along the normal vector .

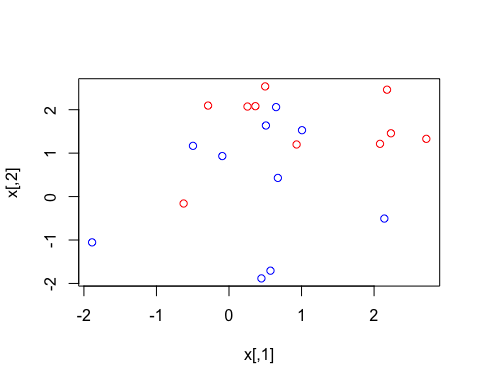
  
*Linear SVM*

# Non-Linear SVM

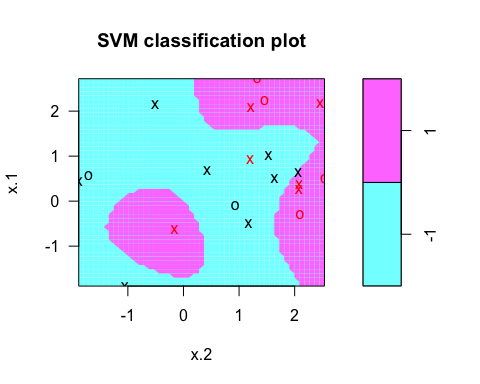
Non-linear classifiers are created by applying the kernel trick then generating maximum-margin hyperplanes.

# Support Vector Machines in R

#Generate random numbers  
x<-matrix(rnorm(40),ncol=2)   
set.seed(333)  
  
#First 10 elements of Y are -1 and the rest are 1  
y<-c(rep(-1,10),rep(1,10))  
  
#Add 1 to the last 10 rows of x  
x[y==1,]<- x[y==1,]+1  
  
#Checking if classes are linearly separable  
plot(x,col=(3-y)) #col= 2 means blue, col= 4 means red



dataframe<-data.frame(x=x,y=as.factor(y))  
  
  
  
#----------------------Training a model on the data ----  
  
svm.fit<- svm(y~.,data=dataframe, kernal="linear",cost=10,scale=FALSE)  
plot(svm.fit,dataframe)



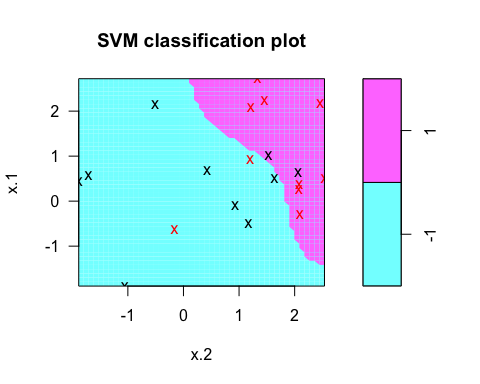
# List my Support Vectors  
svm.fit$index

## [1] 1 3 5 6 7 8 9 10 13 14 15 16 19 20

summary(svm.fit)

##   
## Call:  
## svm(formula = y ~ ., data = dataframe, kernal = "linear", cost = 10,   
## scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 10   
## gamma: 0.5   
##   
## Number of Support Vectors: 14  
##   
## ( 8 6 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## -1 1

# Let's chage Cost  
svm.fit1<- svm(y~.,data=dataframe, kernal="linear",cost=0.1,scale=FALSE)  
plot(svm.fit1,dataframe) # Lower Cost - Wider Margin



#Cross-Validation -Find Best Model for prediction   
tune.out<- tune(svm,y~.,data=dataframe,kernal="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))  
bestmodel=tune.out$best.model  
summary(bestmodel)

##   
## Call:  
## best.tune(method = svm, train.x = y ~ ., data = dataframe, ranges = list(cost = c(0.001,   
## 0.01, 0.1, 1, 5, 10, 100)), kernal = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 5   
## gamma: 0.5   
##   
## Number of Support Vectors: 15  
##   
## ( 8 7 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## -1 1

##----------------------Let's focus on Predicting the Function ---  
  
## Generate Test data  
xtest = matrix(rnorm(40),ncol=2)  
ytest = sample(c(-1,1),20,rep=TRUE)  
xtest[ytest==1,]<- xtest[ytest==1,]+1  
  
testdf<-data.frame(x=xtest,y=as.factor(ytest))  
testdf

## x.1 x.2 y  
## 1 -1.12490028 -0.70278946 -1  
## 2 -0.87430362 0.29986416 -1  
## 3 1.04386162 2.14946968 1  
## 4 -0.58397514 -0.30164149 -1  
## 5 0.17670142 0.45266065 1  
## 6 0.11059280 -0.38540415 -1  
## 7 0.36472778 0.27637203 -1  
## 8 0.05370100 0.15186081 -1  
## 9 2.36867235 0.56861921 1  
## 10 0.61954276 -0.87050006 1  
## 11 -0.70632036 0.07633326 -1  
## 12 1.96687656 0.76964405 1  
## 13 -0.71016648 0.31746484 -1  
## 14 -0.05917365 -1.59268440 -1  
## 15 -0.23094342 -0.18194062 -1  
## 16 1.69200045 2.75614174 1  
## 17 2.35682290 1.11394932 1  
## 18 2.78872284 0.70783492 1  
## 19 1.41071582 2.75409316 1  
## 20 -2.18785470 1.80958455 -1

#Predict testdata(i.e. testdf ) based on bestmodel  
  
ypred=predict(bestmodel,testdf)  
ypred

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20   
## 1 -1 1 1 -1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 1 1 1 -1   
## Levels: -1 1

table(pred=ypred,truth=testdf$y)

## truth  
## pred -1 1  
## -1 9 2  
## 1 2 7

# look only at agreement vs. non-agreement  
# construct a vector of TRUE/FALSE indicating correct/incorrect predictions  
agreement <- ypred == testdf$y  
table(agreement)

## agreement  
## FALSE TRUE   
## 4 16

prop.table(table(agreement))

## agreement  
## FALSE TRUE   
## 0.2 0.8

##-------------- Improving model performance -----  
  
classifier\_rbf <- ksvm(y ~ ., data = testdf, kernel = "rbfdot")  
predictions\_rbf <- predict(classifier\_rbf, testdf)  
predictions\_rbf

## [1] -1 -1 1 -1 -1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 1 1 1 -1  
## Levels: -1 1

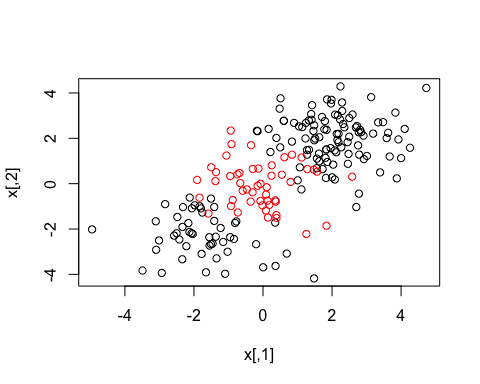
agreement\_rbf <- predictions\_rbf == testdf$y  
table(agreement\_rbf)

## agreement\_rbf  
## FALSE TRUE   
## 2 18

prop.table(table(agreement\_rbf))

## agreement\_rbf  
## FALSE TRUE   
## 0.1 0.9

####-------------------------- RADIAL KERNAL ( play with gamma and cost) ------------------  
  
# -----------Generate random Data----------  
  
set.seed(33)  
x<-matrix(rnorm(400),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)



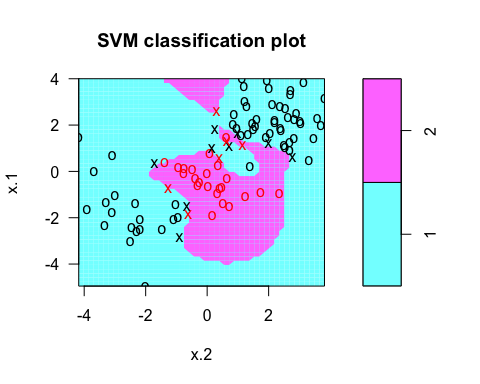
# ----------- Training Model on data --------  
  
train = sample(200,100)  
train

## [1] 61 1 130 56 43 20 158 160 68 122 100 26 55 74 22 3 116  
## [18] 188 174 91 121 153 23 187 142 192 4 185 127 67 155 157 179 125  
## [35] 69 108 200 106 133 49 171 87 135 11 18 76 57 95 190 77 38  
## [52] 72 129 169 175 162 80 196 60 30 177 92 35 176 152 7 126 134  
## [69] 28 199 128 191 173 2 34 58 165 167 86 54 181 145 24 16 156  
## [86] 10 66 112 103 78 31 90 111 123 184 65 53 117 150 89

svmfit1<-svm(y~.,data=dat[train,],kernal="radial",gamma=1,cost=100000)  
svmfit1

##   
## Call:  
## svm(formula = y ~ ., data = dat[train, ], kernal = "radial",   
## gamma = 1, cost = 1e+05)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1e+05   
## gamma: 1   
##   
## Number of Support Vectors: 15

plot(svmfit1,dat[train,])



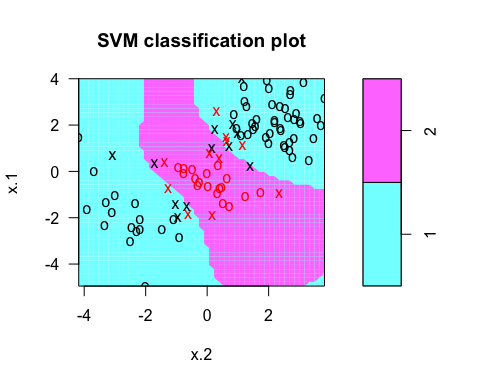
#------------- Cross Validation to set best choice of gamma and cost  
  
tune.out= tune(svm,y~.,data=dat[train,],kernal="radial",ranges=list(cost=c(0.1,10,100,1000)),gamma=c(0.5,1,2,3,4))  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 10  
##   
## - best performance: 0.12   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-01 0.19 0.15951315  
## 2 1e+01 0.12 0.07888106  
## 3 1e+02 0.14 0.10749677  
## 4 1e+03 0.17 0.10593499

bestmodel1<-tune.out$best.model  
bestmodel1

##   
## Call:  
## best.tune(method = svm, train.x = y ~ ., data = dat[train, ],   
## ranges = list(cost = c(0.1, 10, 100, 1000)), kernal = "radial",   
## gamma = c(0.5, 1, 2, 3, 4))  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 10   
## gamma: 0.5 1 2 3 4   
##   
## Number of Support Vectors: 23

plot(bestmodel1,dat[train,])



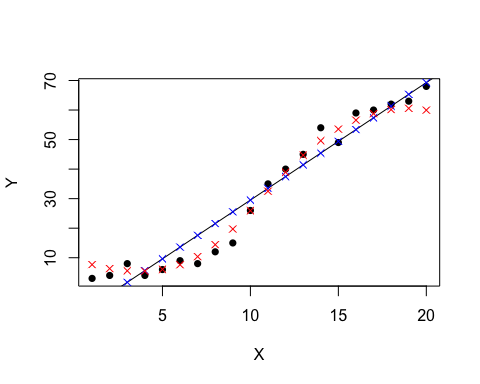
test=-train  
train

## [1] 61 1 130 56 43 20 158 160 68 122 100 26 55 74 22 3 116  
## [18] 188 174 91 121 153 23 187 142 192 4 185 127 67 155 157 179 125  
## [35] 69 108 200 106 133 49 171 87 135 11 18 76 57 95 190 77 38  
## [52] 72 129 169 175 162 80 196 60 30 177 92 35 176 152 7 126 134  
## [69] 28 199 128 191 173 2 34 58 165 167 86 54 181 145 24 16 156  
## [86] 10 66 112 103 78 31 90 111 123 184 65 53 117 150 89

# Prediction Error Comparison between Linear Regression and SVM #  
  
  
# Plot the data  
plot(svm.data, pch=16)  
  
# Create a linear regression model  
lr\_model <- lm(Y ~ X, svm.data)  
  
# Add the fitted line  
abline(lr\_model)  
  
# make a prediction for each X  
predictedY <- predict(lr\_model, svm.data)  
  
# display the predictions  
points(svm.data$X, predictedY, col = "blue", pch=4)  
  
# Function to find the Error  
rmse <- function(error)   
{  
 sqrt(mean(error^2))  
}  
error <- lr\_model$residuals # same as data$Y - predictedY  
predictionRMSE <- rmse(error)  
predictionRMSE

## [1] 5.703778

# Support Vector Machine(Finding root mean square error)  
  
  
  
# Create Support Vector Model  
svm\_model <- svm(Y ~ X , svm.data)  
  
# make a prediction for each X  
predictedY1 <- predict(svm\_model, svm.data)  
  
# display the predictions  
points(svm.data$X, predictedY1, col = "red", pch=4)



# Function to find the Error  
error <- svm.data$Y - predictedY1  
svrPredictionRMSE <- rmse(error)   
  
tuneResult <- tune(svm, Y ~ X, data = svm.data,ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9)))  
print(tuneResult)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## epsilon cost  
## 0 128  
##   
## - best performance: 8.702864

svrPredictionRMSE

## [1] 3.157061

# Resources

* [Support Vector Regression with R](http://www.svm-tutorial.com/2014/10/support-vector-regression-r/)
* [Computing and visualizing LDA in R](https://tgmstat.wordpress.com/2014/01/15/computing-and-visualizing-lda-in-r/)
* [SVM example with Iris Data in R](http://rischanlab.github.io/SVM.html)