R Notebook

#Support Vetor Machines

An introduction to support vector machines taken from lecture notes presented by Dr. Luis Torgo at Dalhousie University on October 22 and 24.

SVM'S were introduced in 1992 at the Conference on Learning Theory in Pittsburgh, Pasadena. Since their inception SVM's have been applied with success in a wide range of areas like: bio-informatics, text mining, and hand-written character recognition.

They are useful in solving binary classification problems where a linear separation of class sets are necessary. They have proven very effective for linearly separable problems.

However, as most scientists know, most data is not linearly separable. To get around this issue we then project our data into a much higer dimensional space where we can then apply a linear approximation.

Map the original data into a new (higher dimensional) coordinate system where our class data becomes linearly separable by a maximum margin hyperplane

#Support Vector Machines in R for Now we will learn how to implement a SVM in R. The package is accessible at e1071: https://cran.r-project.org/web/packages/e1071/index.html .

e1071: Misc Functions of the Department of Statistics, Probability Theory Group. Provides functions for latent class analysis, short time Fourier Transforms, fuzzy clustering, support vector machines, shortest path computation, bagged clustering, and more.

Import the statistical package,

```
library(e1071)
set.seed(1234)
```

##Load the "Iris" dataset.

```
data(iris)
```

```
head(iris)
```

```
##   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
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Display the total number of rows in the dataset.

```
nrow(iris)
```

```
## [1] 150
```

Create our training and testing sets of data.

We use the sample() function to take a random number of rows from our dataset.

```
sampleset <-sample(1:nrow(iris),100)
training<-iris[sampleset,]
test<- iris[-sampleset,]
```

Learning Phase: Apply the SVM model on the training data

```
model1 <- svm(Species ~.,training)
model2 <- svm(Species ~.,training,cost=20)
```

Prediction Phase:

```
ps1<- predict(model1,test)
ps2<- predict(model2,test)
```

##Error calculations: Create a confusion matrix to alert us of when our model made correct / incorrect classifications.

```
mc1 <- table(ps1,test$Species)
mc2 <- table(ps2,test$Species)
```

```
mc1
```

```
##
## ps1          setosa versicolor virginica
##   setosa         18          0         0
##   versicolor      0         17         1
##   virginica       0          1        13
```

```
mc2
```

```
##
## ps2          setosa versicolor virginica
##   setosa         18          0         0
##   versicolor      0         17         1
##   virginica       0          1        13
```

```
e1 <- 1 - sum( diag(mc1) ) / sum(mc1)
e2 <- 1 - sum( diag(mc2) ) / sum(mc2)

print(e1)
```

```
## [1] 0.04
```

```
print(e2)
```

```
## [1] 0.04
```

Support Vector Machines for Regression

We can extend the notion of a Support Vector Machine for vector regression. Our goal is to find a function that has at most an ϵ deviation from the given training cases. This means that we will allow errors smaller than ϵ.

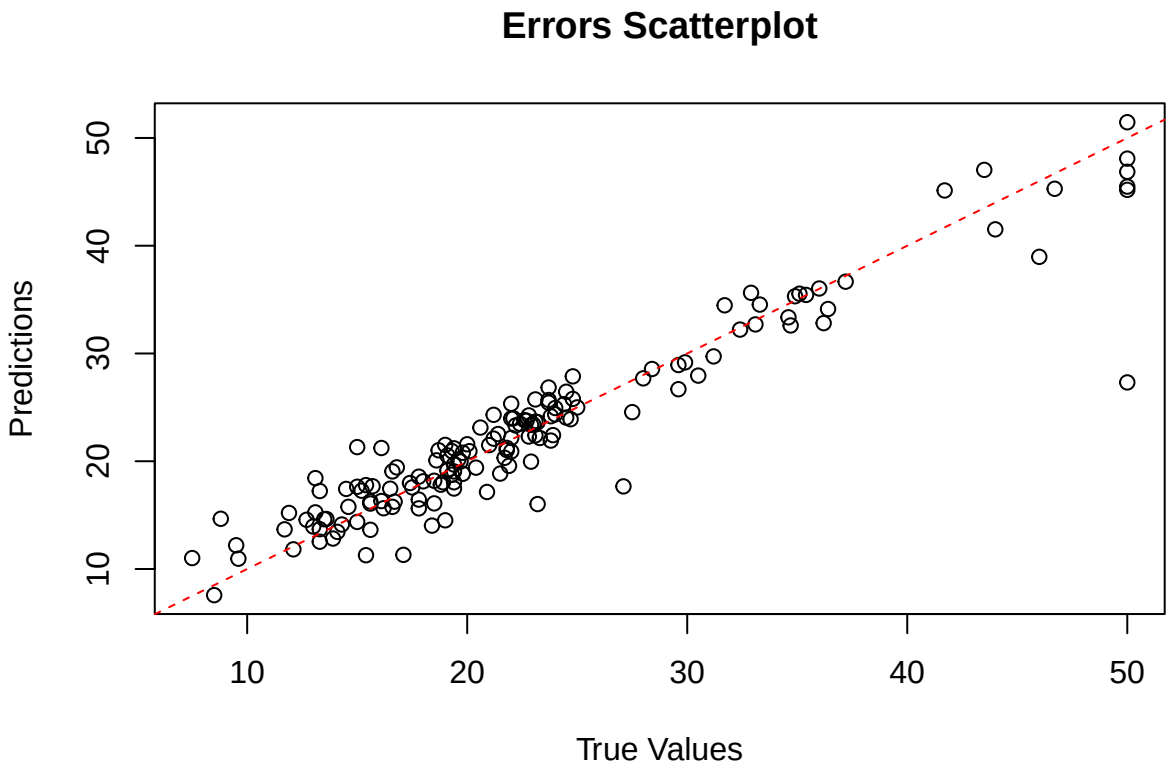
Here we present a sample linear regression:

```
library(e1071)
data(Boston,package='MASS')
set.seed(1234)
#obtain a random sample from our dataset of 354 rows
```

```r
sp <- sample(1:nrow(Boston),354)

#define the training set
tr <- Boston[sp,]
#define the test set
ts <- Boston[-sp,]

#The support vector machine
s <- svm(medv ~ .,tr,cost = 10,epsilon=0.1)

#Make predictions on our data
preds <-predict(s,ts)

#find the mean squared error
mean((ts$medv-preds)^2)
```

```
## [1] 9.087205
```

```r
#Plot our predicted values vs our True values.
plot(ts$medv,preds,main="Errors Scatterplot", ylab='Predictions', xlab='True Values')

#Define a straight line through the origin that categorizes where accurate predictions should lie.
abline(0,1,col='red',lty=2)
```



We can repeat the same procedure as above but loop through our parameter values of ϵ. This will help us determine which value of ϵ is best to fit our model with.

```r
#Define an epsilon range to test
es <- seq(0.5,0.00001, by=-0.0002)
#Create an array to store our error calculations
```

```r
errs<- c()

#Loop through each value of epsilon and calculate the error of the model
for(e in es)
  {
  s<-svm(medv ~ .,tr,cost=10,epsilon=e)
  preds <- predict(s,ts)
  err <-mean((ts$medv-preds)^2)

  errs <- c(errs,err)
  }

plot(es,errs,type="l")
```