svm

Dr.metales

12/14/2019

Introduction

In this paper we will explore the **support vector machine** model using two data sets, the first is **Tiatanic** data to which we will fit a classification **SVM** model, and the second data is **BostonHousing** data (from **mlbench** package) that will be used to fit a regression model.

First let's call the packages that we need for our analysis.

```
library(tidyverse)
library(caret)
library(e1071)
```

SVM model for classification

To show how to use SVM model for classification, we will make use of wellknown titanic data (to import this data click here) where the **Survived** variable will be considered as our target and all the other variables as predictors.Let's call and display some rows from this data

```
data<-read_csv("train.csv")
glimpse(data)
```

```
## Observations: 891
## Variables: 12
## $ PassengerId <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
## $ Survived    <dbl> 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,...
## $ Pclass      <dbl> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3,...
## $ Name        <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bra...
## $ Sex         <chr> "male", "female", "female", "female", "male", "mal...
## $ Age         <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, ...
## $ SibSp       <dbl> 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4,...
## $ Parch       <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1,...
## $ Ticket      <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "1138...
## $ Fare        <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, ...
## $ Cabin       <chr> NA, "C85", NA, "C123", NA, NA, "E46", NA, NA, NA, ...
## $ Embarked    <chr> "S", "C", "S", "S", "S", "Q", "S", "S", "S", "C", ...
```

As we can see some variables cannot be used as predictors since thay have unique valaues such as **PassengerId**,**Name**,**Ticket**, and **Cabin**, hence we remove these variables.

```
data<-data[,-c(1,4,9,11)]
glimpse(data)
```

```
## Observations: 891
## Variables: 8
## $ Survived <dbl> 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,...
```

```
## $ Pclass   <dbl> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, 2,...
## $ Sex      <chr> "male", "female", "female", "female", "male", "male",...
## $ Age      <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, 39,...
## $ SibSp    <dbl> 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 0,...
## $ Parch    <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0,...
## $ Fare     <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, 51....
## $ Embarked <chr> "S", "C", "S", "S", "S", "Q", "S", "S", "S", "C", "S"...
```

Some variables should be treated as factors than characters such as **Sex** and **Embarked**, or than numeric such as **Survived** and **Pclass**, we convert thus, these variables to factor type using the function **mutate_at** from **dplyr** package.

```
data <- data %>%
  mutate_at(c(1,2,3,8),funs(as.factor))
glimpse(data)
```

```
## Observations: 891
## Variables: 8
## $ Survived <fct> 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,...
## $ Pclass   <fct> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, 2,...
## $ Sex      <fct> male, female, female, female, male, male, male, male,...
## $ Age      <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, 39,...
## $ SibSp    <dbl> 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 0,...
## $ Parch    <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0,...
## $ Fare     <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, 51....
## $ Embarked <fct> S, C, S, S, S, Q, S, S, S, C, S, S, S, S, S, S, Q, S,...
```

If we look at the summary of this data we see that the **Age** variable has large number of missing values (177), and also 2 missing values for **Embarked** variable. Since we have a large number of missing values, it will be a very bad idea to remove all the rows that have these values with all the informations that are related to. therfore, we will instead impute these values with the super valuable help of **mice** package. But we do not do this now since we have not yet split the data between training set and test set. If we do we would contaminate the training set during the imputation process since the missing values of the training set may be imputed using some cases from the test set, and this will violate the crutial principle in machine learning that the test set should never be seen by the model during the training process.

```
summary(data)
```

```
##  Survived Pclass      Sex           Age            SibSp
##  0:549    1:216   female:314   Min.   : 0.42   Min.   :0.000
##  1:342    2:184   male  :577   1st Qu.:20.12   1st Qu.:0.000
##           3:491                 Median :28.00   Median :0.000
##                                 Mean   :29.70   Mean   :0.523
##                                 3rd Qu.:38.00   3rd Qu.:1.000
##                                 Max.   :80.00   Max.   :8.000
##                                 NA's   :177
##      Parch            Fare        Embarked
##  Min.   :0.0000   Min.   :  0.00   C   :168
##  1st Qu.:0.0000   1st Qu.:  7.91   Q   : 77
##  Median :0.0000   Median : 14.45   S   :644
##  Mean   :0.3816   Mean   : 32.20   NA's:  2
##  3rd Qu.:0.0000   3rd Qu.: 31.00
##  Max.   :6.0000   Max.   :512.33
##
```

Split the data

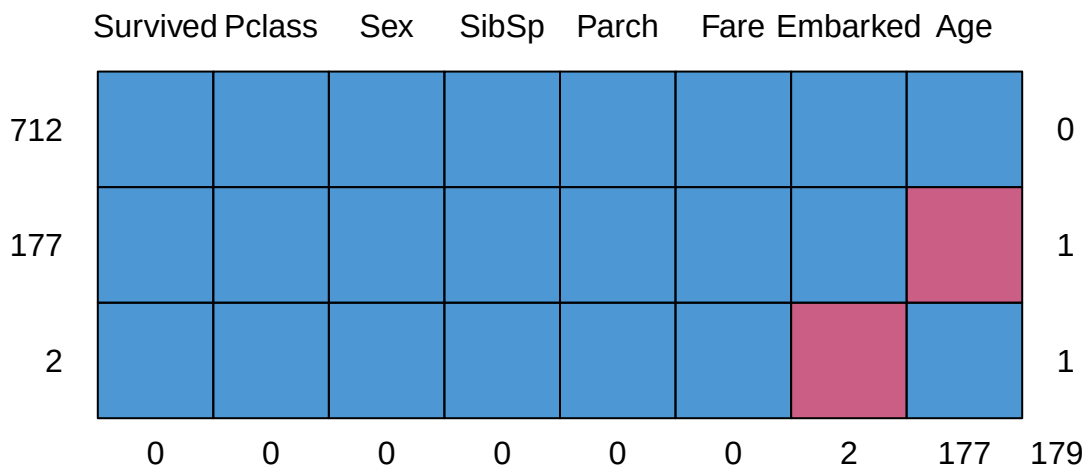
Using **caret** package function **creatDataPartition** to split the data into training set with 80% of the total cases and the remaining 20% to the test set.

```
set.seed(1234)
index<-createDataPartition(data$Survived,p=0.8,list=FALSE)
train<-data[index,]
test<-data[-index,]
```

Before imputing the missing values let's check the distribution of these values in the original set.

Imputation of missing values

```
library(mice)
md.pattern(data)
```




```
##     Survived Pclass Sex SibSp Parch Fare Embarked Age
## 712        1      1   1     1     1    1        1   1   0
## 177        1      1   1     1     1    1        1   0   1
## 2          1      1   1     1     1    1        0   1   1
##            0      0   0     0     0    0        2 177 179
```

As we see we have 179 missing values in total, the **Age** variable has alone 177 and **Embarked** has only 2 values. For the rows we have 712 rows without any missing value, 177 rows with one missing value from the **Age** variable in each and 2 rows with missing values from **Embarked** variable.

No we are ready to imput the missing values separatley for the train and for the test. the **cart** in the argument method is a tree based model used to predict the missing vlues using all the other variables as predictors. The **mice** function uses the markov chain monte carlo algorithm that is why we use seed number for reproducibility purpose.

```
imput<-mice(train,m=1,method = "cart",maxit=20 ,seed=1111,print=FALSE)
train<-complete(imput)

imput1<-mice(test,m=1,method = "cart",maxit=20 ,seed=1111,print=FALSE)
test<-complete(imput1)
```

It is time now to train our model, at first step we use the default values of the svm model.

Train the model

```
set.seed(123)
modelsvm<-svm(Survived~., data=train)
summary(modelsvm)
```

```
##
## Call:
## svm(formula = Survived ~ ., data = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  337
##
##  ( 170 167 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

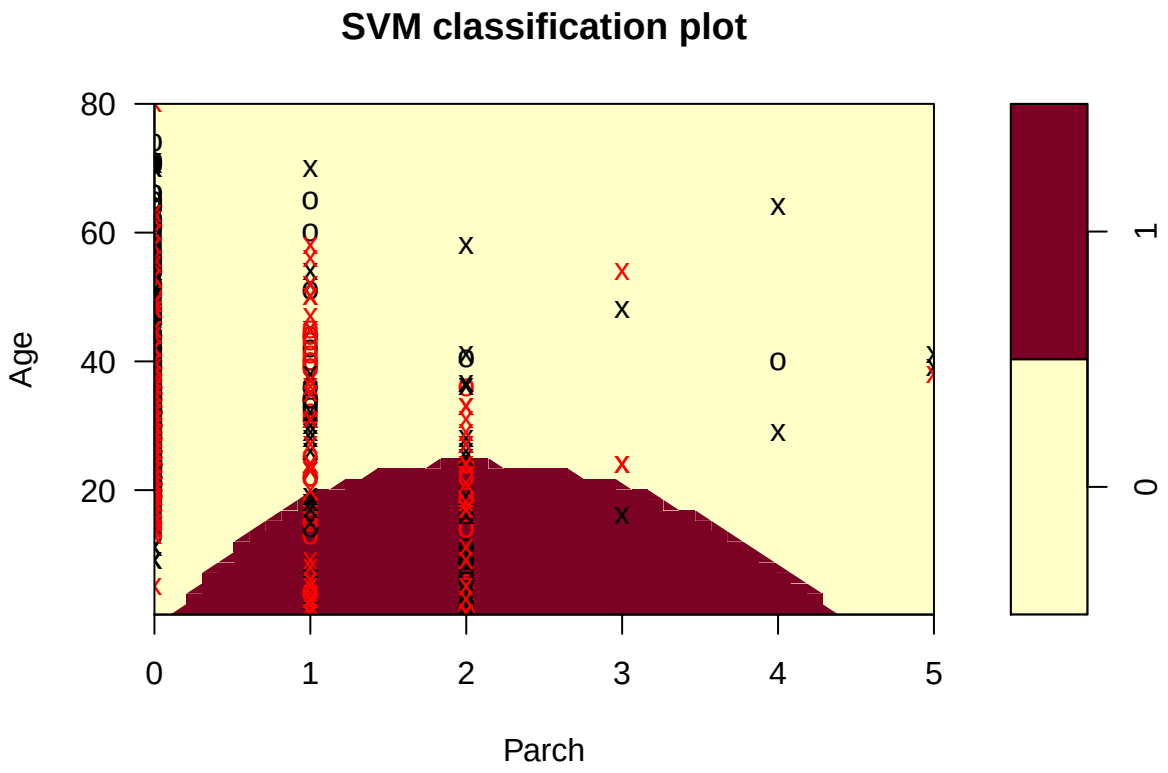
the model with itself knows the type of the dependent variable and decide which type of model we will use. Here it has used classification type. the default kernel basis function is **Radial**, but there exist other function in the **e1071** package such as **linear**, **polynomial** or **sigmoid**. The **cost** parameter (it acts as regularization term) is by default set to 1 . You can check the detail of this model by simply runing **?svm**.

For our data this model perform 148 support vectors for class 0 and 145 for class 1 (293 in total).

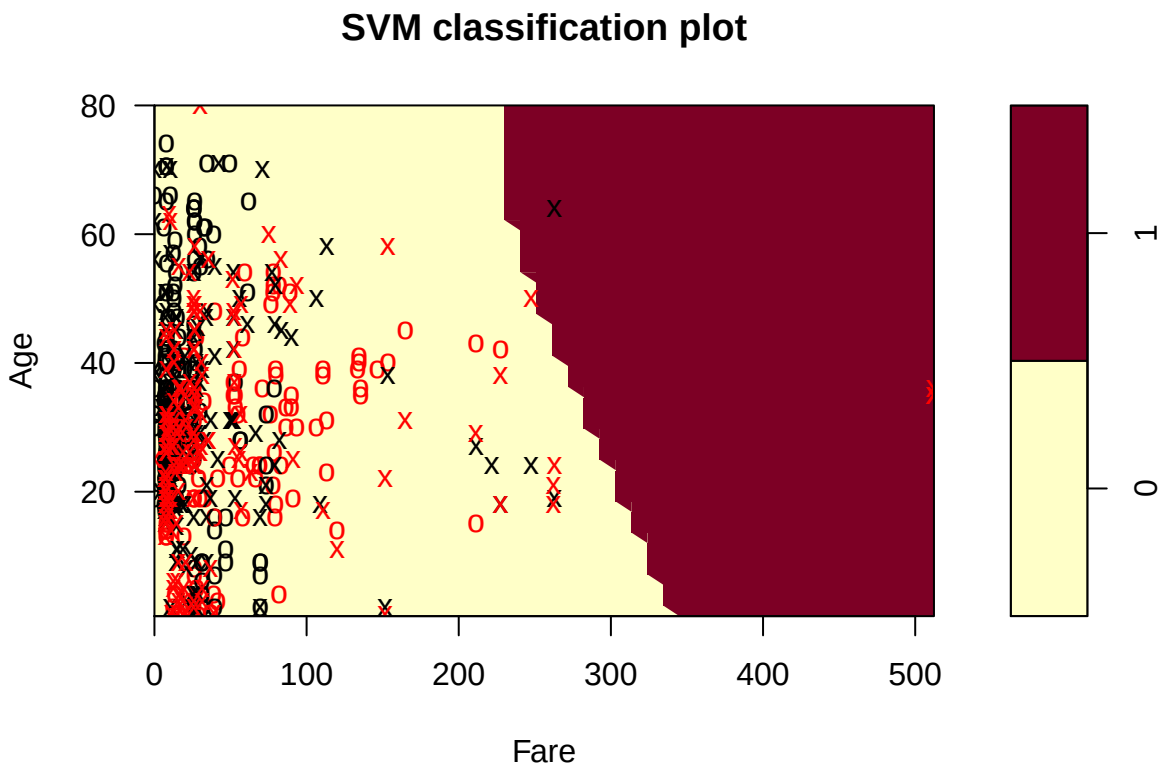
Let's plot this model to get a first glance at the separated regions. this poor representation does not mean poor model, it only means that the representation in the two dimension space is poor. this happens,because

we use a plane with less important predictors such as **Fare** and we cannot plot the important predictors since they are of factor type. But the separation in the high dimension may be good. however, we can easly check later on the model performence by the accuracy rate.

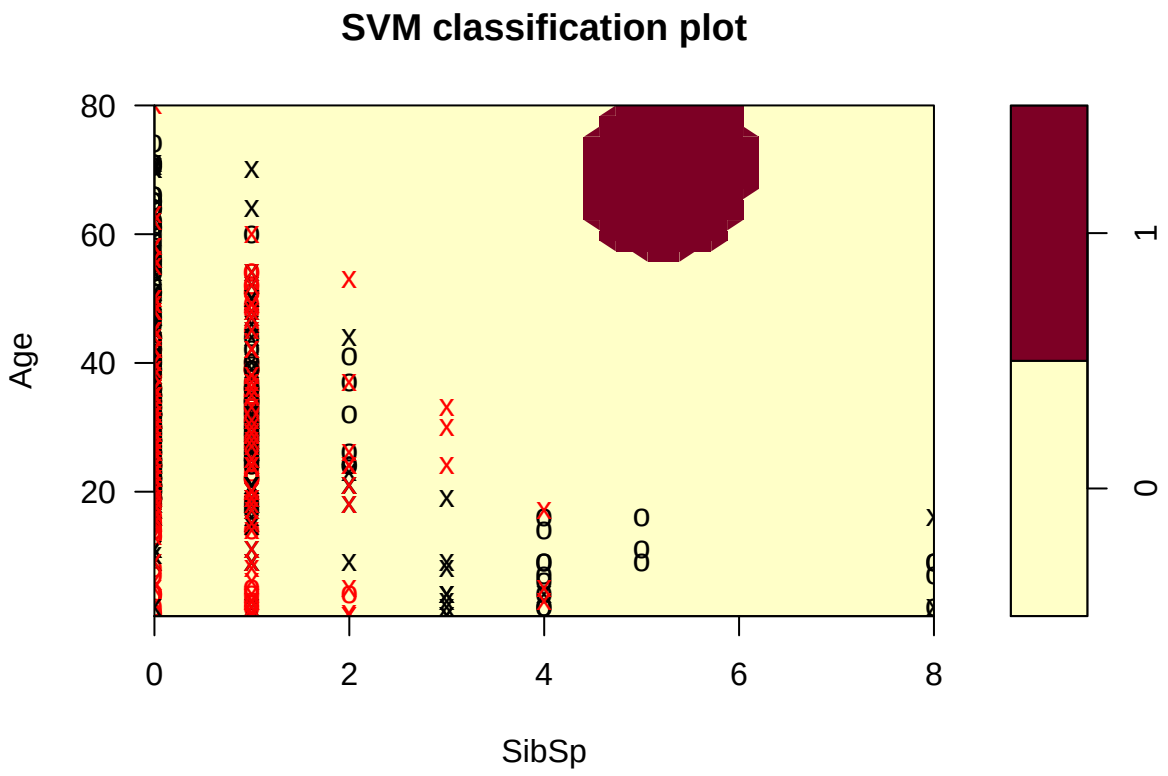
```
plot(modelsvm,data=train,
     Age~Parch,
     slice =list(Sex="male",Pclass=1,SibSp=1,Fare=53,                    Embarked="C"))
```




```
plot(modelsvm,data=train,
     Age~Fare,
     slice =list(Sex="male",Pclass=1,SibSp=1,Parch=0,                    Embarked="C"))
```




```
plot(modelsvm,data=train,
     Age~SibSp,
     slice =list(Sex="male",Pclass=1,Fare=53,Parch=0,                    Embarked="C"))
```



Now let's predict our model using the training set and also create the confusion matrix

Model evaluation and prediction

```
pred<-predict(modelsvm,train)
confusionMatrix(as.factor(pred),as.factor(train$Survived))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 393  63
##          1  47 211
##
##                Accuracy : 0.8459
##                  95% CI : (0.8173, 0.8716)
##     No Information Rate : 0.6162
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6706
##
##  Mcnemar's Test P-Value : 0.1527
##
##             Sensitivity : 0.8932
```

```
##             Specificity : 0.7701
##          Pos Pred Value : 0.8618
##          Neg Pred Value : 0.8178
##              Prevalence : 0.6162
##          Detection Rate : 0.5504
##    Detection Prevalence : 0.6387
##       Balanced Accuracy : 0.8316
##
##        'Positive' Class : 0
##
```

This model with the default values gives a good accuracy rate about 84.59% for the training set, but let's check this model using the test set.

```
pred1<-predict(modelsvm,test)
confusionMatrix(as.factor(pred1),as.factor(test$Survived))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 100  27
##          1   9  41
##
##                Accuracy : 0.7966
##                  95% CI : (0.7297, 0.8533)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 1.87e-07
##
##                   Kappa : 0.5476
##
##  Mcnemar's Test P-Value : 0.004607
##
##             Sensitivity : 0.9174
##             Specificity : 0.6029
##          Pos Pred Value : 0.7874
##          Neg Pred Value : 0.8200
##              Prevalence : 0.6158
##          Detection Rate : 0.5650
##    Detection Prevalence : 0.7175
##       Balanced Accuracy : 0.7602
##
##        'Positive' Class : 0
##
```

The accuracy rate using the test set is about roughly 80%.

Hyperparameters optimization

we can go back and try to fine tuning our model to get some improvment. we can try different values for cost function and epsilon.

```r
set.seed(123)
modeltuned <- tune(svm, Survived~.,data=train,
                   ranges = list(epsilon=c(0,.5,1),                cost=c(0.1,50,90)))
summary(modeltuned)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  epsilon cost
##        0   50
##
## - best performance: 0.1653169
##
## - Detailed performance results:
##   epsilon cost     error dispersion
## 1     0.0  0.1 0.1848592 0.04143248
## 2     0.5  0.1 0.1848592 0.04143248
## 3     1.0  0.1 0.1848592 0.04143248
## 4     0.0 50.0 0.1653169 0.04331267
## 5     0.5 50.0 0.1653169 0.04331267
## 6     1.0 50.0 0.1653169 0.04331267
## 7     0.0 90.0 0.1680947 0.04853653
## 8     0.5 90.0 0.1680947 0.04853653
## 9     1.0 90.0 0.1680947 0.04853653
```

we see that the lowest error rate is about **16.53%** (83,47% accuracy rate) under 0 for epsilon and 50 for cost. Even this rate is lower than the one that we have got earlier with the default values , but let's go ahead and check the accuracy rate for the test set using the new parameters values

Let's include this new parameter and check wether we get any improvment using the test set.

```r
set.seed(123)
modelsvm<-svm(Survived~., data=train,cost=50, epsilon=0)
pred2<-predict(modelsvm,test)
confusionMatrix(as.factor(pred2),as.factor(test$Survived))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 99 25
##          1 10 43
##
##                Accuracy : 0.8023
##                  95% CI : (0.7359, 0.8582)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 7.432e-08
##
##                   Kappa : 0.564
##
```

```
##  Mcnemar's Test P-Value : 0.01796
##
##             Sensitivity : 0.9083
##             Specificity : 0.6324
##          Pos Pred Value : 0.7984
##          Neg Pred Value : 0.8113
##              Prevalence : 0.6158
##          Detection Rate : 0.5593
##    Detection Prevalence : 0.7006
##       Balanced Accuracy : 0.7703
##
##        'Positive' Class : 0
##
```

we get a slight improvment from 79.66% to 80.23%.

Another posible way to improve the model is to try another kernel basis function, and since we deal with binary classification model let's use **sigmoid** funcion with the new parameter values.

```
set.seed(123)
modelsvm<-svm(Survived~., data=train , kernel= "sigmoid",cost=50, epsilon=0)
pred2<-predict(modelsvm,test)
confusionMatrix(as.factor(pred2),as.factor(test$Survived))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 90 21
##          1 19 47
##
##                Accuracy : 0.774
##                  95% CI : (0.7052, 0.8334)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 5.385e-06
##
##                   Kappa : 0.5197
##
##  Mcnemar's Test P-Value : 0.8744
##
##             Sensitivity : 0.8257
##             Specificity : 0.6912
##          Pos Pred Value : 0.8108
##          Neg Pred Value : 0.7121
##              Prevalence : 0.6158
##          Detection Rate : 0.5085
##    Detection Prevalence : 0.6271
##       Balanced Accuracy : 0.7584
##
##        'Positive' Class : 0
##
```

As we see **sigmoid** kernel is a bad choice since the accuracy rate has come down from 80.23% to 77.4%. in practice the **Radial** basis function works well with classification type.

SVM model for regression

SVM model can also be used to predict continous variables. For this we will predict the **medv** the price median value of homes in USA , for more detail about this data run **?BostonHousing**.

```
library(mlbench)
data("BostonHousing")
glimpse(BostonHousing)
```

```
## Observations: 506
## Variables: 14
## $ crim    <dbl> 0.00632, 0.02731, 0.02729, 0.03237, 0.06905, 0.02985, ...
## $ zn      <dbl> 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.5, 12.5, 12.5, 12.5,...
## $ indus   <dbl> 2.31, 7.07, 7.07, 2.18, 2.18, 2.18, 7.87, 7.87, 7.87, ...
## $ chas    <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ nox     <dbl> 0.538, 0.469, 0.469, 0.458, 0.458, 0.458, 0.524, 0.524...
## $ rm      <dbl> 6.575, 6.421, 7.185, 6.998, 7.147, 6.430, 6.012, 6.172...
## $ age     <dbl> 65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96.1, 100.0,...
## $ dis     <dbl> 4.0900, 4.9671, 4.9671, 6.0622, 6.0622, 6.0622, 5.5605...
## $ rad     <dbl> 1, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, ...
## $ tax     <dbl> 296, 242, 242, 222, 222, 222, 311, 311, 311, 311, 311,...
## $ ptratio <dbl> 15.3, 17.8, 17.8, 18.7, 18.7, 18.7, 15.2, 15.2, 15.2, ...
## $ b       <dbl> 396.90, 396.90, 392.83, 394.63, 396.90, 394.12, 395.60...
## $ lstat   <dbl> 4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19.15, 29.9...
## $ medv    <dbl> 24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, ...
```

Let's check the summary of this data

```
summary(BostonHousing)
```

```
##       crim                zn              indus       chas
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   0:471
##  1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1: 35
##  Median : 0.25651   Median :  0.00   Median : 9.69
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74
##       nox               rm              age              dis
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127
##       rad              tax           ptratio            b
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90
##      lstat            medv
##  Min.   : 1.73   Min.   : 5.00
```

```
##  1st Qu.: 6.95   1st Qu.:17.02
##  Median :11.36   Median :21.20
##  Mean   :12.65   Mean   :22.53
##  3rd Qu.:16.95   3rd Qu.:25.00
##  Max.   :37.97   Max.   :50.00
```

Fortunatley we do not have any missing value, but the variable values are of of different magnitude, so to make the trainingprocess more faster we recommend to normalize or standardize the predictors first. To do this we use **preprocess** function from **caret** package. But before that let's split the data into training set and test set.

```
set.seed(1234)
index<-sample(nrow(BostonHousing),size = floor(0.8*(nrow(BostonHousing))))
train<-BostonHousing[index,]
test<-BostonHousing[-index,]
```

And now we center and scale the variables for both the **train** and **test** set after removing the target variable

```
scaled<-preProcess(train[,-14],method=c("center","scale"))
trainscaled<-predict(scaled,train)
testscaled<-predict(scaled,test)
```

Now we print the dimension of all the sets to be sure everything is fine.

```
dim(BostonHousing)
```

```
## [1] 506  14
```

```
dim(trainscaled)
```

```
## [1] 404  14
```

```
dim(testscaled)
```

```
## [1] 102  14
```

We are ready now to train our model.

```
set.seed(123)
modelsvmR<-svm(medv~., data=trainscaled )
summary(modelsvmR)
```

```
##
## Call:
## svm(formula = medv ~ ., data = trainscaled)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
```

```
##        cost:  1
##       gamma:  0.07142857
##     epsilon:  0.1
##
##
## Number of Support Vectors:  276
```

Then we get the prediction and the root mean squared error **RMSE** as follows.

```
pred3<-predict(modelsvmR,test)
head(pred3)
```

```
##        1        5        7        8        9       11
## 22.15377 22.15377 22.15377 22.15377 22.15377 22.15377
```

```
RMSE(pred3,test$medv)
```

```
## [1] 8.99958
```

As we did above we can finetuning our parameters or changing the kernel finction to looking for some possible improvment.