SVM Training and Testing Models in R

**1. Objective**

In This[**R**](http://data-flair.training/blogs/r-programming-tutorial/)blog, we are going to discuss the SVM training and testing models in R. We will also discuss the most important Package**e1071.** Functions of e1071 i.e. *svm(), predict(), plot(), tune()* to execute SVM in R.

[Get the best R books to become a master in R Programming.](http://data-flair.training/blogs/r-best-books-to-learn-r/)

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/svm-training-and-testing-models-in-r.jpg)

**2. R – SVM Training and Testing Models**

There are several [**packages**](http://data-flair.training/blogs/r-packages-tutorial/) to execute [**SVM**](http://data-flair.training/blogs/svm-support-vector-machine-tutorial/) in R. The first and most intuitive package is the **e1071** package.

**The e1071 Package –**

This package was the first implementation of SVM in R.

The **svm() function** in e1071 provides a rigid interface to libsvm. By using visualization and parameter tuning methods. Refer some of the features of libsvm library given below:

* Offers quick and easy implementation of SVMs.
* Provides most common[**kernels**](http://data-flair.training/blogs/svm-kernel-functions/), including linear, polynomial, RBF, and sigmoid.
* Offers computation power for decision and probability values for predictions. Also provides class weighting in the classification mode, and cross-validation.

First, you need to set the path to include the directory where the e1071 package is. Then you have to install and include it.

You can use > ?svm to see the help information of the interface.

Install e1071 package and load using the following commands:

1. install.packages (‘e1071’, dependencies = TRUE)
2. library(e1071)

The **R** implementation depends on the S3 class mechanisms. It provides a training function with standard and formula interfaces, and a **predict()** method. Also provides a **plot()** method for visualizing data, support vectors, and decision boundaries. We can do Hyperparameter tuning by using the**tune()** framework. It performs a grid search over specified parameter ranges.

**3. Main Functions in the e1071 Package**

The main functions in the e1071 package are:

* **svm() –** Used to train SVM.
* **predict() –** Using this method obtains predictions from the model, as well as decision values from the binary classifiers.
* **plot() –** Visualizing data, support vectors, and decision boundaries if provided.
* **tune() –** Hyperparameter tuning uses tune() to perform a grid search over specified parameter ranges.

**3.1. The svm() Function**

The**svm()** function trains an SVM. It can do general[**regression**](http://data-flair.training/blogs/r-linear-regression-tutorial/) and classification, as well as density-estimation. Provides a formula interface.

The below data describes some import parameters of the svm() function:

**a) Data –** Specifies an optional data frame that contains the variables present in a model. When you use this parameter, then you do not need to use the x and y parameters. Take the variables by default from the environment which ‘svm’ is called from.

* X – a data matrix, a vector, or a sparse matrix (object of class Matrix provided by the Matrix package). It represents the instances of the dataset and their respective properties. In a data matrix- rows represent the instances, columns represent the properties

**b) Type –** We can use svm as a classification machine, regression machine, or for novelty detection. depending on whether y is a factor or not. The default setting for type is C-classification or eps-regression. It may be overwritten by setting an explicit value. Valid options are:

* C-classification
* nu-classification
* one-classification (for novelty detection)
* eps-regression
* nu-regression
* degree

**c) parameter –** It requires for the kernel of type polynomial (default: 3)

* gamma – parameter needed for all kernels except linear (default: 1/(data dimension))
* coef0 – parameter needed for kernels of type polynomial and sigmoid (default: 0)
* cost – the cost of constraints violation (default: 1)—it is the ‘C’-constant of the regularization term in the Lagrange formulation.

**3.2. The plot() Function**

Use the **plot() function** to view the built model with a scatter plot of the input. It optionally draws a filled contour plot of the class regions. plot() function used to represent data, support vectors and models in a visual form. How to use this function:

plot.svm(x, data, **for**mula, fill = TRUE, grid = 50, slice = list(),symbolPalette = palette(), svSymbol = "x", dataSymbol = "o", ...)

[](https://data-flair.training/blogs/r-interview-questions-and-answers/)

Here,

* **x** – An object of class svm.
* **Formula** – Formula selecting the visualized two dimensions. Only needed when we use more than two input variables.
* **Fill** – Switch indicating whether a contour plot for the class regions should add.
* **Grid** – Granularity for the contour plot.
* **Slice** – A list of named numeric values for the dimensions held constant. If dimensions not specified, we can fix it at 0.
* **Model** – Represents an object of class svm data, which results from the svm() function.
* **Data** – Represents the data to visualize. It should use the same data used for building the model in the svm() function.
* **symbolPalette** – Color palette used for the class the data points and support vectors belong to.
* **svSymbol** – Symbol used for support vectors.
* **dataSymbo** – Symbol used for data points (other than support vectors).
* svm allows a simple graphical visualization of classification models.

**3.3. The predict() Function**

The**predict()** **function** predicts values based on a model derived by an SVM. It returns the class labels in case of classification with a class membership value or the decision values of the classifier. It also returns a vector of predicted labels for a classification problem.

Following are the steps to execute the predict() function:

**Step 1:** Divide the dataset into a training set and a test set. We can do it by using below commands:

1. Index <- 1:nrow(cats)
2. Testindex <- sample(index,trunk(length(index)/3))
3. Testset <- cats[Testindex, ]
4. Trainset <- cats[-testindex, ]

**Step 2:** Run the model again and predict classes by using the training set. Use commands as below:

1. Model <-svm(Sex~., data=trainset)
2. Prediction <- predict (model, testset[-1])

**Step 3:** Generate the confusion matrix by cross-tabulating the true and predicted values

Tab <- table(pred=prediction, true=testset[,1])

The confusion matrix is a tabular layout. It represents the performance of a[**supervised learning**](http://data-flair.training/blogs/learning-rules-in-neural-network/)algorithm in the graphical form. In a confusion matrix, each column represents instances by the predicted class. On the other hand, each row of the matrix represents the instance of the actual class.

**[Test Your R Knowledge](https://data-flair.training/blogs/category/quiz/r-quiz/" \t "_blank)**

**3.4. The Tune() Function**

It tunes hyper parameters of statistical methods using a grid search over supplied parameter ranges.

Below is represented how to use this function:

1. tune(method, train.x, train.y = NULL, data = **list**(), validation.x = NULL, validation.y = NULL, ranges = NULL, predict.func = predict, tunecontrol = tune.control(), ...)
2. best.tune(...)

* **Method –** It is the function to be tuned or a character string naming such a function.
* **x –** It is a formula or a matrix of predictors.
* **y –** It is the response variable if train.x is a predictor matrix. It is ignored if train.x is a formula.
* **Data** – It is the data, when a formula interface is used. It is ignored, if predictor matrix and response are supplied directly.
* **x –** It is an optional validation set. The response can be included in validation.x or separately specified using validation.y depending on whether a formula interface is used or not.
* **y –** It is only used for bootstrap and fixed validation set (see tune.control)
* **ranges** – It is a named list of parameter vectors spanning the sampling space. The vectors will usually be created by seq.
* **func** – It is optional, when the standard predict behavior is inadequate.
* **Tunecontrol** – It is object of class “tune.control”, as created by the function tune.control(). when omitted, then tune.control() gives the defaults.
* **…** – Further parameters passed to the training functions.