SVM Classifier implementation in R

For SVM classifier implementation in R programming language using caret package, we are going to examine a tidy dataset of Heart Disease. Our motive is to predict whether a patient is having heart disease or not.

To work on big datasets, we can directly use some machine learning packages. Developer community of R programming language has built some great packages to make our work easier. The beauty of these packages is that they are well optimized and can handle maximum exceptions to make our job simple, we just need to call functions for implementing algorithms with the right parameters.

The principle behind an SVM classifier (Support Vector Machine) algorithm is to build a hyperplane separating data for different classes. This hyperplane building procedure varies and is the main task of an SVM classifier. The main focus while drawing the hyperplane is on maximizing the distance from hyperplane to the nearest data point of either class. These nearest data points are known as Support Vectors.

Caret Package Installation

The R programming machine learning caret package(Classification And Regression Training) holds tons of functions that helps to build predictive models. It holds tools for data splitting, pre-processing, feature selection, tuning and supervised – unsupervised learning algorithms, etc.

Heart Disease Recognition Data Set Description

Heart Disease data set consists of 14 attributes data. All the attributes consist of numeric values. First 13 variables will be used for predicting 14th variables. The target variable is at index 14.

Heart Disease Recognition Problem Statement

To model a classifier for predicting whether a patient is suffering from any heart disease or not.

SVM classifier implementation in R with Caret Package

R caret Library and Data Import

For implementing SVM in r, we only need to import caret package, just past the below command in R console to import r machine learning package Caret.

For importing the data and manipulating it, we are going to use data frames. First of all, we need to download the dataset. You can download the dataset our repository. It's a CSV file i.e, Comma Separated Values file. All the data values are separated by commas. After downloading the CSV file, you need to set your working directory via console else save the data file in your current working directory.

For importing data into an R data frame, we can use read.csv() method with parameters as a file name and whether our dataset consists of the 1st row with a header or not. If a header row exists then, the header should be set TRUE else header should set to FALSE.

For checking the structure of data frame we can call the function str() over heart_df:

The output shows us that our dataset consists of 300 observations each with 14 attributes.

To check top 5-6 rows of the dataset, we can use head().

The Range of values of the attributes are different but all attributes consist of numeric data.

giving better results as compared to Linear classifier even after tuning it. It may be due to overfitting

Data Slicing

Data slicing is a step to split data into train and test set. Training data set can be used specifically for our model building. Test dataset should not be mixed up while building model. Even during standardization, we should not standardize our test set.

The set.seed() method is used to make our work replicable. As we want our readers to learn concepts by coding these snippets. To make our answers replicable, we need to set a seed value. During partitioning of data, it splits randomly but if our readers will pass the same value in the set.seed() method. Then we both will get identical results.

The caret package provides a method createDataPartition() for partitioning our data into train and test set. We are passing 3 parameters. The "y" parameter takes the value of variable according to which data needs to be partitioned. In our case, target variable is at V14, so we are passing heart_df$V14 (heart data frame's V14 column).

The "p" parameter holds a decimal value in the range of 0-1. It's to show that percentage of the split. We are using p=0.7. It means that data split should be done in 70:30 ratio. The "list" parameter is for whether to return a list or matrix. We are passing FALSE for not returning a list. The createDataPartition() method is returning a matrix "intrain" with record's indices.

By passing values of intrain, we are splitting training data and testing data.

The line training <- heart_df[intrain,] is for putting the data from data frame to training data. Remaining data is saved in the testing data frame, testing <- heart_df[-intrain,].

For checking the dimensions of our training data frame and testing data frame, we can use these.

Preprocessing & Training

Preprocessing is all about correcting the problems in data before building a machine learning model using that data. Problems can be of many types like missing values, attributes with a different range, etc. To check whether our data contains missing values or not, we can use anyNA() method. Here, NA means Not Available.

Since it's returning FALSE, it means we don't have any missing values.

Dataset summarized details

For checking the summarized details of our data, we can use summary() method. It will give us a basic idea about our dataset's attributes range; it shows us that all the attributes have a different range. So, we need to standardize our data. We can standardize data using caret's preProcess() method.

Our target variable consists of 2 values 0, 1. It should be a categorical variable. To convert these to categorical variables, we can convert them to factors.

This code will convert training data frame's "V14" column to factor variable.

Training the SVM model

Caret package provides train() method for training our data for various algorithms. We just need to pass different parameter values for different algorithms. Before train() method, we will first use trainControl() method. It controls the computational nuances of the train() method.

We are setting 3 parameters of trainControl() method. The "method" parameter holds the details about resampling method. We can set "method" with many values like "boot", "boot632", "cv", "repeatedcv", "LOOCV", "LGOCV" etc. For this tutorial, let's try to use repeatedcv i.e, repeated cross-validation.

The "number" parameter holds the number of resampling iterations. The "repeats " parameter contains the complete sets of folds to compute for our repeated cross-validation. We are using setting number =10 and repeats =3. This trainControl() methods returns a list. We are going to pass this on our train() method.

Before training our SVM classifier, set.seed().

For training SVM classifier, train() method should be passed with "method" parameter as "svmLinear". We are passing our target variable V14. The "V14~." denotes a formula for using all attributes in our classifier and V14 as the target variable. The "trControl" parameter should be passed with results from our trainControl() method. The "preProcess" parameter is for preprocessing our training data.

As discussed earlier for our data, preprocessing is a mandatory task. We are passing 2 values in our "preProcess" parameter "center" & "scale". These two help for centering and scaling the data. After preProcessing these convert our training data with mean value as approximately "0" and standard deviation as "1". The "tuneLength" parameter holds an integer value. This is for tuning our algorithm.

Trained SVM model result

You can check the result of our train() method. We are saving its results in a svm_Linear variable.

It's a linear model therefore, it just tested at value "C" =1.

Test Set Prediction

Now, our model is trained with C value as 1. We are ready to predict classes for our test set. We can use predict() method.

The caret package provides predict() method for predicting results. We are passing 2 arguments. Its first parameter is our trained model and second parameter "newdata" holds our testing data frame. The predict() method returns a list, we are saving it in a test_pred variable.

How Accurately our model is working?

Using confusion matrix, we can print statistics of our results. It shows that our model accuracy for test set is 86.67%.

By following the above procedure, we can build our svmLinear classifier.

We can also do some customizations for selecting C value(Cost) in Linear classifier. This can be done by inputting values in grid search. The next code snippet will show you, building & tuning of an SVM classifier with different values of C. We are going to put some values of C using expand.grid() into "grid" data frame. Next step is to use this data frame for testing our classifier at specific C values. It needs to be put in train() method with tuneGrid parameter.

The plot is showing that our classifier is giving best accuracy on C = 0.05. Let's try to make predictions using this model for our test set.
Let's check its accuracy using confusion -matrix.
The results of confusion matrix show that this time the accuracy on the test set is 87.78 %.

SVM Classifier using Non-Linear Kernel

In this section, we will try to build a model using Non-Linear Kernel like Radial Basis Function. For using RBF kernel, we just need to change our train() method's "method" parameter to "svmRadial". In Radial kernel, it needs to select proper value of Cost "C" parameter and "sigma" parameter.
It's showing that final sigma parameter's value is 0.04744793 & C parameter's value as 0.25. Let's try to test our model's accuracy on our test set. For predicting, we will use predict() with model's parameters as svm_Radial & newdata= testing.
We are getting an accuracy of 87.78%. So, in this case with values of C=0.25 & sigma= 0.04744793, we are getting good results.
Let's try to test & tune our classifier with different values of C & sigma. We will use grid search to implement this.
grid_radial dataframe will hold values of sigma & C. Value of grid_radial will be given to train() method's tuneGrid parameter.
Awesome, we ran our SVM-RBF kernel. It calculated variations and gave us best values of sigma & C. It's telling us that best values of sigma= 0.025 & C=1 Let's check our trained models' accuracy on the test set.
For our svm_Radial_Grid classifier, it's giving an accuracy of 86.67%. So, it shows Radial classifier is not giving better results as compared to Linear classifier even after tuning it. It may be due to overfitting.