

Support Vector Machines (SVMs) for classification

Freddy Hernández

22 de octubre de 2019



Support Vector Machines (SVMs)

SVMs were developed by Cortes & Vapnik (1995) for binary classification.

Url: <https://link.springer.com/article/10.1023/A:1022627411411>



Machine Learning, 20, 273–297 (1995)

© 1995 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Support-Vector Networks

CORINNA CORTES
VLADIMIR VAPNIK
AT&T Bell Labs., Holmdel, NJ 07733, USA

corinna@neural.att.com
vlad@neural.att.com

Editor: Lorenza Saitta

Abstract. The *support-vector network* is a new learning machine for two-group classification problems. The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensures high generalization ability of the learning machine. The idea behind the support-vector network was previously implemented for the restricted case where the training data can be separated without errors. We here extend this result to non-separable training data.

High generalization ability of support-vector networks utilizing polynomial input transformations is demonstrated. We also compare the performance of the support-vector network to various classical learning algorithms that all took part in a benchmark study of Optical Character Recognition.

Keywords: pattern recognition, efficient learning algorithms, neural networks, radial basis function classifiers, polynomial classifiers.



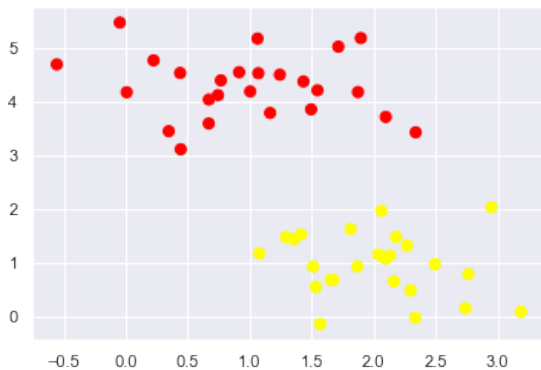
An overview of Support Vector Machines

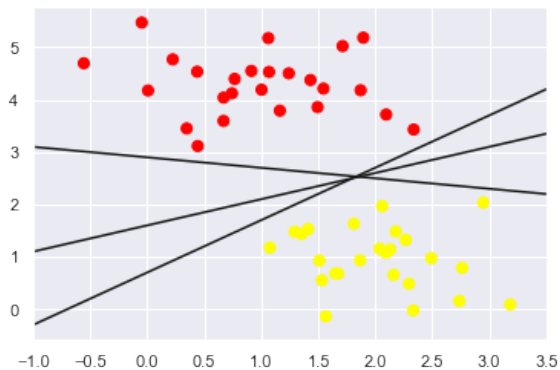
<https://www.svm-tutorial.com/2017/02/svms-overview-support-vector-machines/>

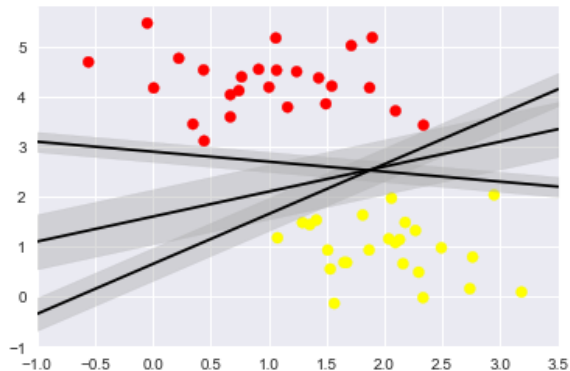
http://jermmy.xyz/images/2017-12-23/support_vector_machines_succinctly.pdf

https://rpubs.com/Joaquin_AR/267926



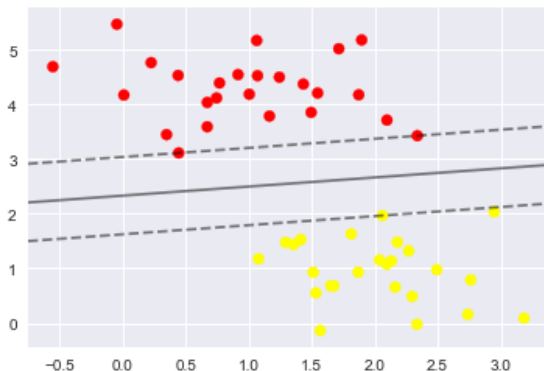




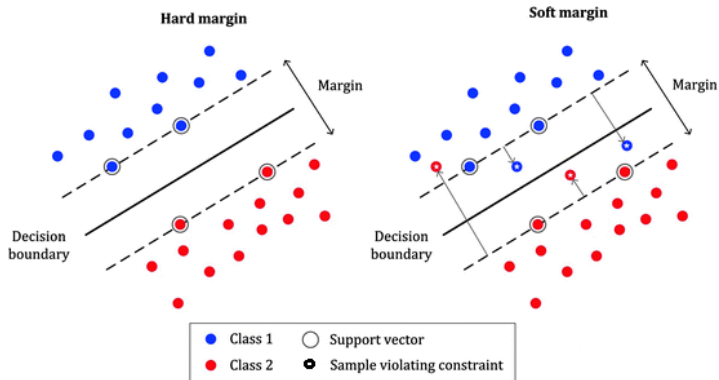


Maximal margin hyperplane (hard)

The objective of the Support Vector Machine algorithm is to find the hyperplane that has the maximum margin in an N -dimensional space that distinctly classifies the data points.



Support Vector Classifier o Soft Margin SVM



- ① The first implementation of SVM in R (R Development Core Team 2005) was introduced in the **e1071** package. [Visit this url.](#)
- ② Package **kernlab** features a variety of kernel-based methods and includes a SVM method based on the optimizers used in libsvm and bsvm. [Visit this url.](#)
- ③ Package **klaR** includes an interface to SVMlight, a popular SVM implementation that additionally offers classification tools such as Regularized Discriminant Analysis. [Visit this url.](#)
- ④ package **svmpath** provides an algorithm that fits the entire path of the SVM solution. [Visit this url.](#)

To install all packages:

```
install.packages(c("e1071", "kernlab", "klaR", "svmpath"))
```



The package **e1071**

The package **e1071** contains the `svm` function used for SVMs. To install the package:

```
install.packages("e1071")
```

To load the package:

```
library(e1071)
```

The main function is:

```
svm(formula, data, scale=TRUE,  
     kernel=linear or polynomial or radial or sigmoid,  
     degree=3, gamma=1/n, coef0=0, cost=1, ...)
```

Consult the vignette:

<https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf>



Example 1

Let's first download some Train data.

```
url <- "https://raw.githubusercontent.com/rdaymedellin/  
tutoriales_Rday_2019/master/Machine%20Learning/blobs_train.txt"  
Train <- read.table(url, sep=";", header=TRUE)
```



Example 1

Exploring the dimensions and the content of Train dataset.

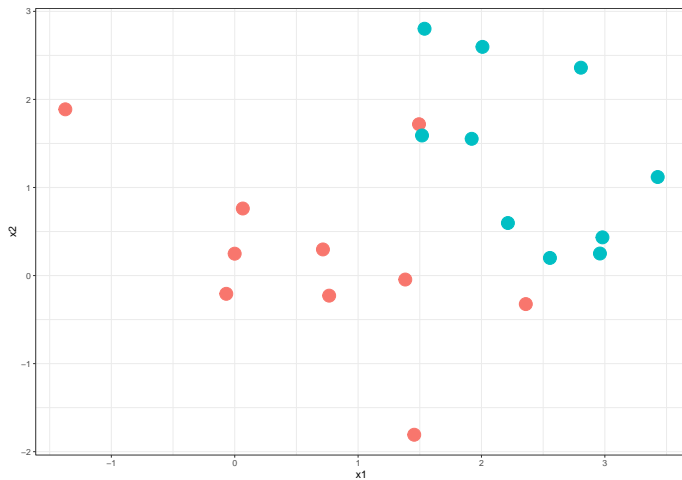
Train

##		x1	x2	y
## 1	1.3823344467	-0.04521314	Pop 1	
## 2	1.4549117978	-1.80768300	Pop 1	
## 3	2.3593089766	-0.32395860	Pop 1	
## 4	0.0654694412	0.76132991	Pop 1	
## 5	-0.0684902073	-0.20724920	Pop 1	
## 6	0.7149229213	0.29658101	Pop 1	
## 7	1.4937061136	1.71847498	Pop 1	
## 8	0.7654051404	-0.22819454	Pop 1	
## 9	-0.0004792237	0.24840834	Pop 1	
## 10	-1.3730953800	1.88776773	Pop 1	
## 11	2.5548979647	0.20013380	Pop 2	
## 12	2.9806430286	0.43380847	Pop 2	
## 13	3.4284307071	1.11900141	Pop 2	
## 14	1.5385761783	2.80283948	Pop 2	
## 15	1.9206093671	1.55289452	Pop 2	
## 16	2.9603100109	0.25015729	Pop 2	
## 17	2.8058915544	2.36046840	Pop 2	
## 18	1.5180460582	1.59061301	Pop 2	
## 19	2.0079858232	2.59699179	Pop 2	
## 20	2.2142667078	0.59660335	Pop 2	



Example 1

```
library(ggplot2)
ggplot(data = Train, aes(x = x1, y = x2, color = as.factor(y))) +
  geom_point(size = 6) + theme_bw() +
  theme(legend.position = "none")
```



Example 1

Here the kernel is linear, scale equals FALSE.

```
svm_lin <- svm(y ~ x1 + x2, data=Train, kernel="linear", scale=FALSE)
```



Example 1

The summary.

```
summary(svm_lin)
```

```
##
## Call:
## svm(formula = y ~ x1 + x2, data = Train, kernel = "linear", scale = FALSE)
##
##
## Parameters:
##   SVM-Type:  C-classification
##   SVM-Kernel: linear
##           cost:  1
##
## Number of Support Vectors:  6
##
##   ( 3 3 )
##
##
## Number of Classes:  2
##
## Levels:
##   Pop 1 Pop 2
```



Example 1

Observations inside margins.

```
svm_lin$index
```

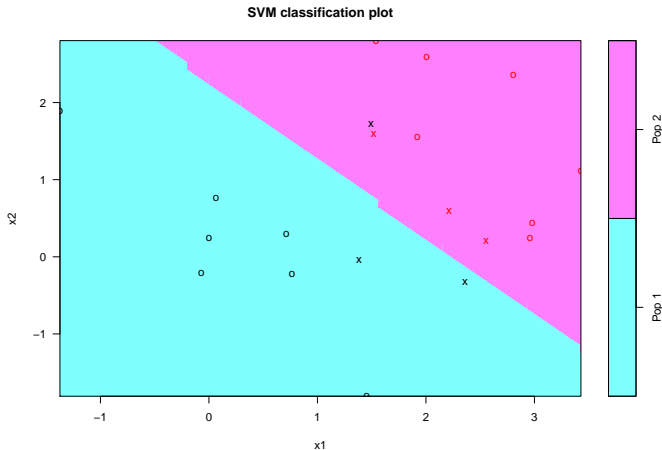
```
## [1]  1  3  7 11 18 20
```



Example 1

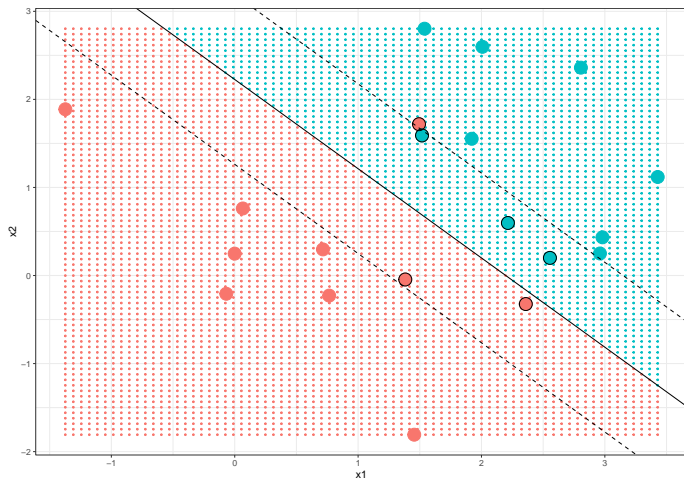
There's a plot function for SVM that shows the decision boundary.

```
plot(x=svm_lin, data=Train, formula=x2~x1,  
     color.palette = cm.colors, grid=50,  
     symbolPalette=c('black', 'red'))
```



Example 1

```
url <- "https://tinyurl.com/y5maxs9r"  
source(url)  
plot_margins(model=svm_lin, data=Train)
```



Example 1

To classify observations we can use the predict function.

```
pred_lin <- predict(object=svm_lin, newdata=Train[, -3])
```

To obtain the [Confusion Matrix](#) we can use:

```
tabla <- table(Predictions=pred_lin, True=Train$y)
tabla
```

```
##           True
## Predictions Pop 1 Pop 2
##      Pop 1      9      0
##      Pop 2      1     10
```

To obtain the accuracy we can use:

```
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.95
```



Example 1

Download the Test dataset. In this example the Test dataset has many observations than Train dataset, it is rare.

```
url <- "https://raw.githubusercontent.com/rdaymedellin/  
tutoriales_Rday_2019/master/Machine%20Learning/blobs_test.txt"  
Test <- read.table(url, sep=";", header=TRUE)
```

Exploring the dimensions and the content of Train dataset.

```
dim(Test)
```

```
## [1] 100  3
```

```
Test[49:52, ] # Four middle observations
```

```
##           x1           x2      y  
## 49 -1.4132094 -0.7593428 Pop 1  
## 50 -1.3746188 -0.4218237 Pop 1  
## 51  2.0752031  1.2803257 Pop 2  
## 52  0.9922126  0.8767329 Pop 2
```



Example 1

To classify new observations using the information in the Test dataset we can use the `predict` function.

```
pred_lin <- predict(object=svm_lin, newdata=Test)
```

To obtain the **Confusion Matrix** we can use:

```
tabla <- table(Predictions=pred_lin, True=Test$y)
tabla
```

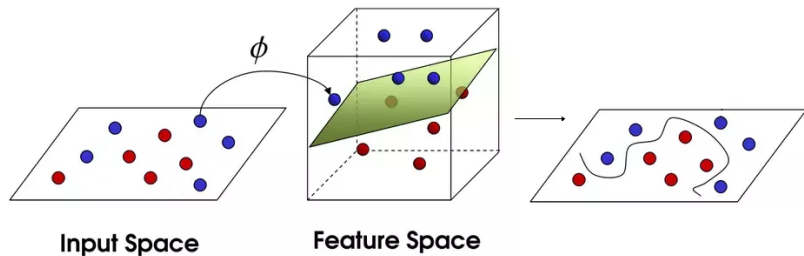
```
##           True
## Predictions Pop 1 Pop 2
##      Pop 1    47     8
##      Pop 2     3    42
```

To obtain the accuracy we can use:

```
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.89
```





See the next video:

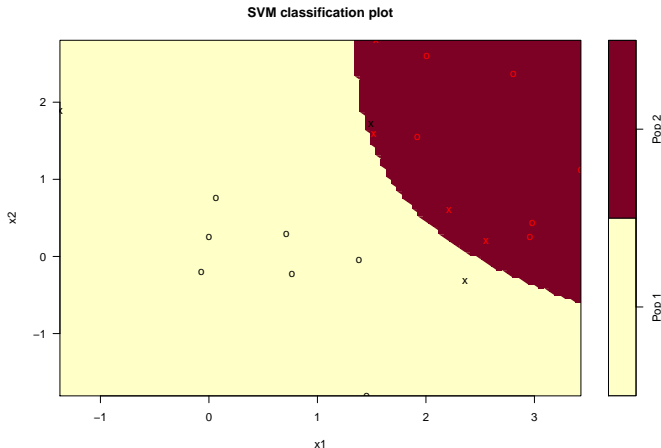
<https://www.youtube.com/watch?v=3liCbRZPrZA>

- linear
- polynomial
- radial
- sigmoid



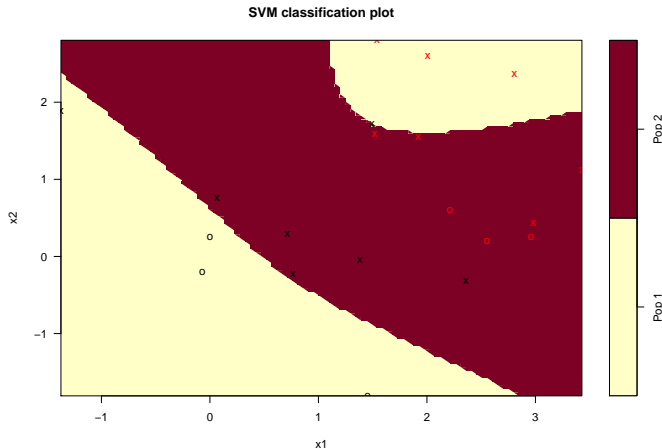
Example 1

```
svm_pol <- svm(y ~ ., data=Train, kernel="polynomial", scale=FALSE)
plot(x=svm_pol, data=Train, formula=x2~x1, grid=100)
```



Example 1

```
svm_sig <- svm(y ~ ., data=Train, kernel="sigmoid", scale=FALSE)  
plot(x=svm_sig, data=Train, formula=x2~x1, grid=100)
```



Example 1

Accuracy for each model.

```
pred <- predict(object=svm_lin, newdata=Test)
tabla <- table(pred, Test$y)
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.89
```

```
pred <- predict(object=svm_pol, newdata=Test)
tabla <- table(pred, Test$y)
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.83
```

```
pred <- predict(object=svm_sig, newdata=Test)
tabla <- table(pred, Test$y)
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.58
```



What is inside svm object?

What is the class of svm_lin?

```
class(svm_lin)
```

```
## [1] "svm.formula" "svm"
```

What is inside svm_lin?

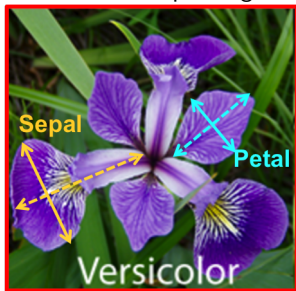
```
names(svm_lin)
```

```
## [1] "call"           "type"           "kernel"
## [4] "cost"           "degree"         "gamma"
## [7] "coef0"          "nu"             "epsilon"
## [10] "sparse"         "scaled"         "x.scale"
## [13] "y.scale"        "nclasses"       "levels"
## [16] "tot.nSV"        "nSV"            "labels"
## [19] "SV"             "index"          "rho"
## [22] "compprob"       "probA"          "probB"
## [25] "sigma"          "coefs"          "na.action"
## [28] "fitted"         "decision.values" "terms"
```



Example with iris

Could be used sepal length and sepal width to predict the Species?



```
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



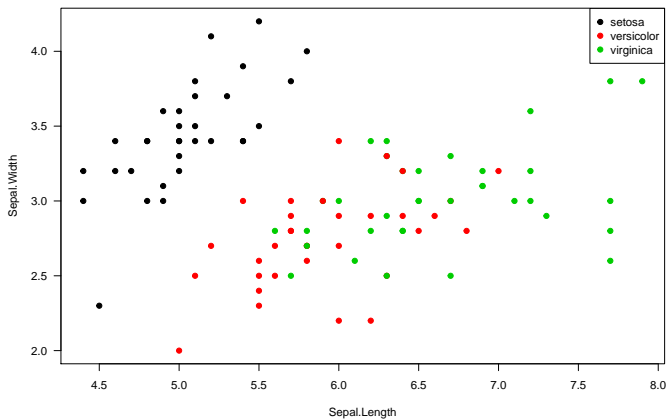
Example iris

We can split the original data into Train and Test.

```
indices <- sample(1:150, size=100)
Train <- iris[indices, c(1, 2, 5)]
Test  <- iris[-indices, c(1, 2, 5)]
```

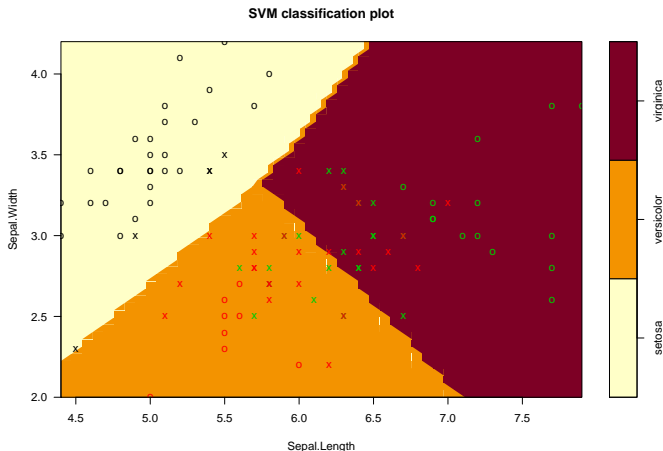


Example iris



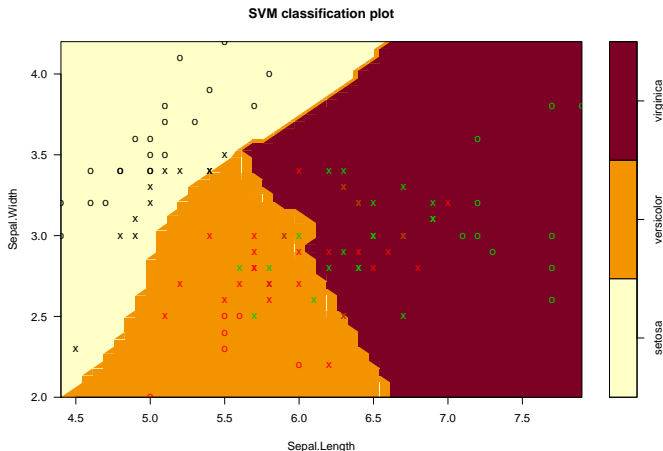
Example iris

```
iris1 <- svm(Species ~ Sepal.Width + Sepal.Length,  
             data=Train, kernel="linear", scale=TRUE)  
plot(iris1, Train)
```



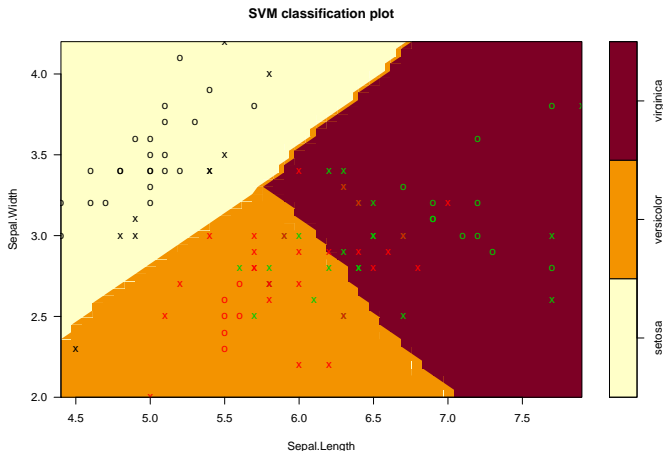
Example iris

```
iris2 <- svm(Species ~ Sepal.Width + Sepal.Length,  
             data=Train, kernel="polynomial", scale=TRUE)  
plot(iris2, Train)
```



Example iris

```
iris3 <- svm(Species ~ Sepal.Width + Sepal.Length,  
             data=Train, kernel="radial", scale=TRUE)  
plot(iris3, Train)
```



Example iris

```
iris4 <- svm(Species ~ Sepal.Width + Sepal.Length,  
             data=Train, kernel="sigmoid", scale=TRUE)  
plot(iris4, Train)
```



Example iris

```
pred <- predict(object=iris1, newdata=Test)
tabla <- table(pred, Test$Species)
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.8
```

```
pred <- predict(object=iris2, newdata=Test)
tabla <- table(pred, Test$Species)
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.76
```

```
pred <- predict(object=iris3, newdata=Test)
tabla <- table(pred, Test$Species)
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.8
```

```
pred <- predict(object=iris4, newdata=Test)
tabla <- table(pred, Test$Species)
sum(diag(tabla)) / sum(tabla)
```

```
## [1] 0.72
```



- degree: parameter needed for kernel of type polynomial (default: 3).
- gamma: parameter needed for all kernels except linear (default: $1/(\text{data dimension})$).
- cost: cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.
- gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.



Pros and Cons associated with SVM

Pros:

- It works really well with clear margin of separation.
- It is effective in high dimensional spaces.
- It is effective in cases where number of dimensions is greater than the number of samples.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Cons:

- It doesn't perform well, when we have large data set because the required training time is higher.
- It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping.
- SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

