

Overview

- Explanation of support vector machine (SVM), a popular [machine learning](#) algorithm or classification
- Implementation of SVM in R and Python
- Learn about the pros and cons of Support Vector Machines(SVM) and its different applications

Introduction

Mastering [machine learning algorithms](#) isn't a myth at all. Most beginners start by learning regression. It is simple to learn and use, but does that solve our purpose? Of course not! Because you can do so much more than just Regression!

Think of machine learning algorithms as an armory packed with axes, swords, blades, bows, daggers, etc. You have various tools, but you ought to learn to use them at the right time. As an analogy, think of 'Regression' as a sword capable of slicing and dicing data efficiently, but incapable of dealing with highly complex data. On the contrary, 'Support Vector Machines' is like a sharp knife – it works on smaller datasets, but on complex ones, it can be much stronger and powerful in building machine learning models.

By now, I hope you've now mastered [Random Forest](#), [Naive Bayes Algorithm](#), and [Ensemble Modeling](#). If not, I'd suggest you take out a few minutes and read about them as well. In this article, I shall guide you through the basics to advanced knowledge of a crucial [machine learning](#) algorithm, [support vector machines](#).

You can learn about Support Vector Machines in course format here (it's free!):

- [Support Vector Machines \(SVM\) in Python and R](#)

If you're a beginner looking to start your data science journey, you've come to the right place! Check out the below comprehensive courses, curated by industry experts, that we have created just for you:

- [Introduction to Data Science](#)
- [Machine Learning Certification Course for Beginners](#)
- [Certified AI & ML Blackbelt+ Program](#)

Understanding Support Vector Machine algorithm from examples (along with code)

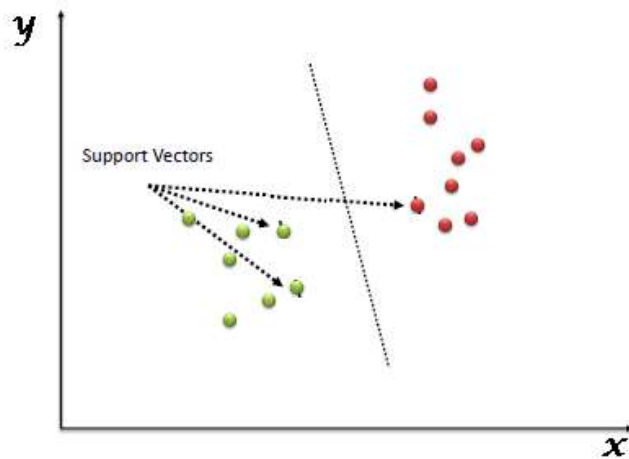
Table of Contents

1. What is Support Vector Machine?
2. How does it work?
3. How to implement SVM in Python and R?
4. How to tune Parameters of SVM?
5. Pros and Cons associated with SVM

What is the Support Vector Machine?

“Support Vector Machine” (SVM) is a supervised [machine learning algorithm](#) that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-

plane that differentiates the two classes very well (look at the below snapshot).



Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/line).

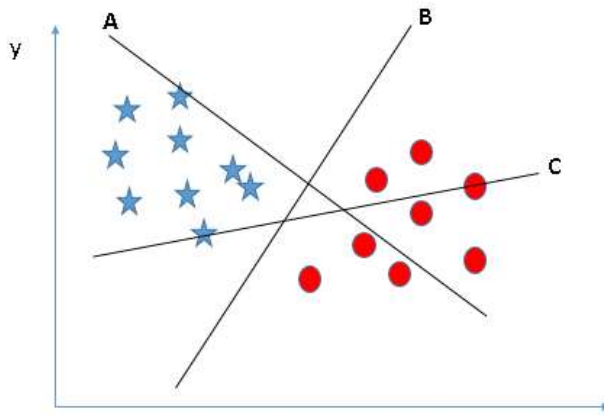
You can look at [support vector machines](#) and a few examples of their working here.

How does it work?

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is “How can we identify the right hyper-plane?”. Don’t worry, it’s not as hard as you think!

Let’s understand:

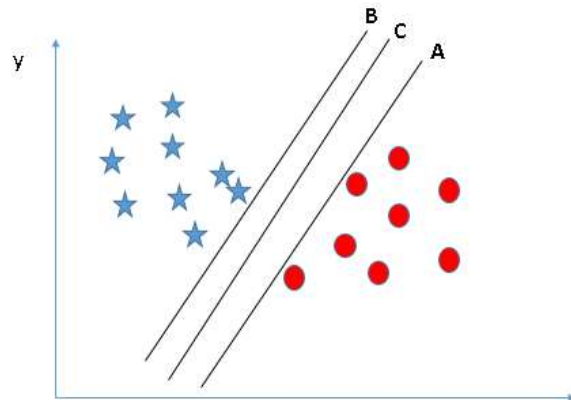
- **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B, and C). Now, identify the right hyper-plane to classify stars and circles.



× You need to

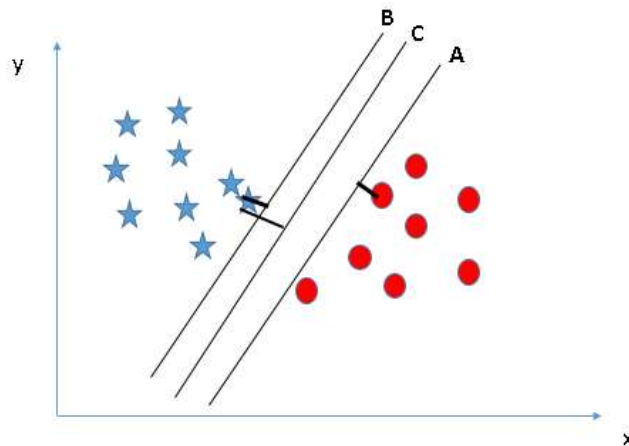
remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.

- **Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B, and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?



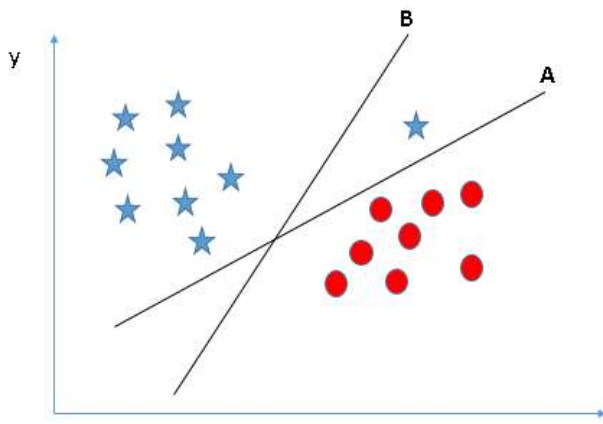
× Here, maximizing the

distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let's look at the below snapshot:



Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

- **Identify the right hyper-plane (Scenario-3):**Hint: Use the rules as discussed in previous section to identify the right hyper-plane

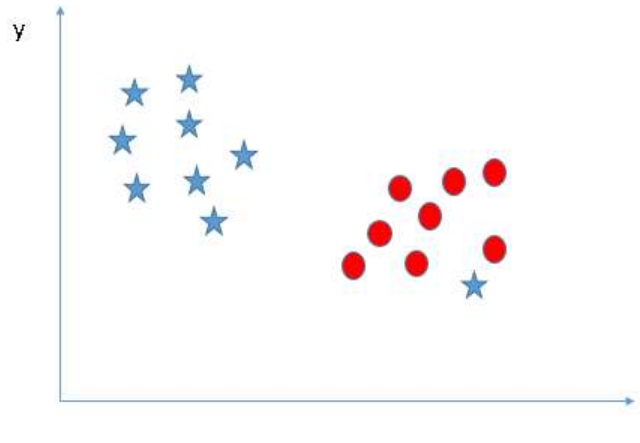


× Some of you may have

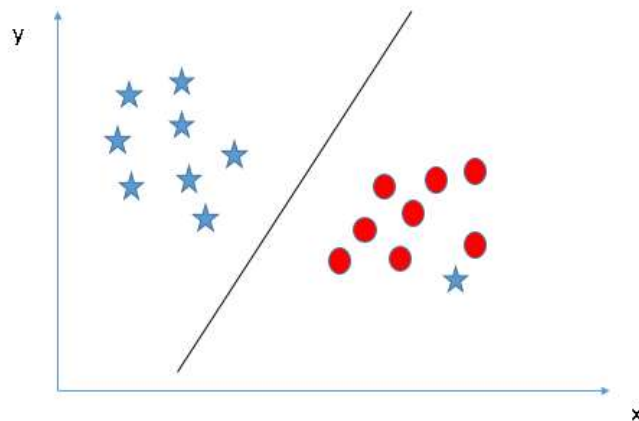
selected the hyper-plane **B** as it has higher margin compared to **A**. But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a

classification error and A has classified all correctly. Therefore, the right hyper-plane is A.

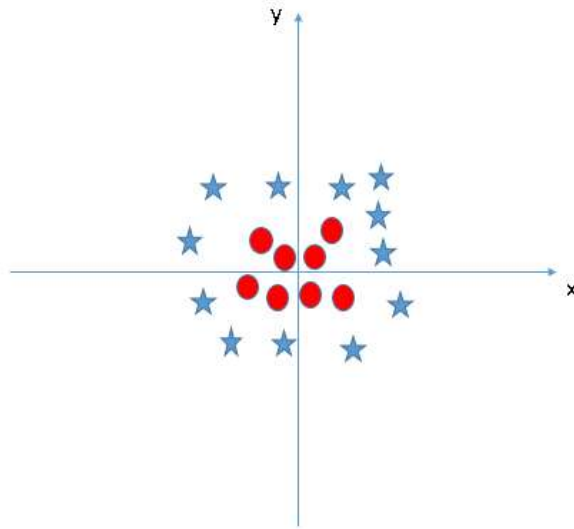
- **Can we classify two classes (Scenario-4)?:** Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of other(circle) class as an outlier.



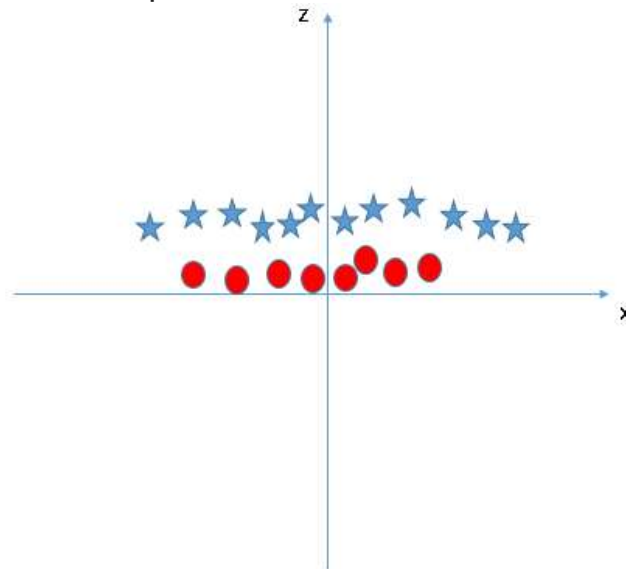
As I have already mentioned, one star at other end is like an outlier for star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.



- **Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature $z = x^2 + y^2$. Now, let's plot the data points on axis x and z :



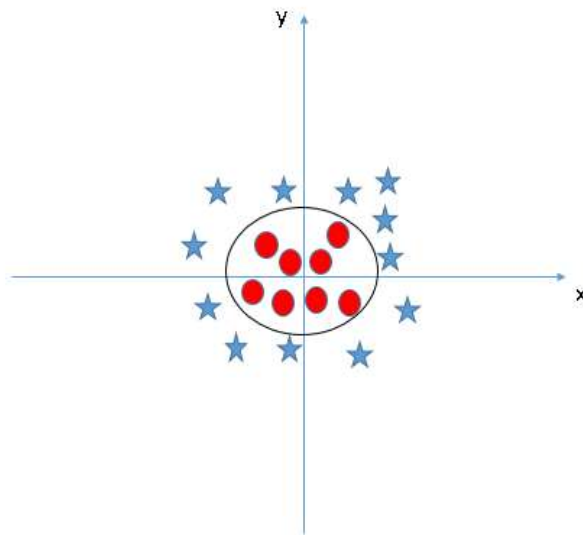
In above plot, points to consider are:

- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z .

In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should

we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the **kernel trick**. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:



Now, let's look at the methods to apply SVM classifier algorithm in a data science challenge.

You can also learn about the working of Support Vector Machine in video format from this [Machine Learning Certification](#).

How to implement SVM in Python and R?

In Python, scikit-learn is a widely used library for implementing [machine learning](#) algorithms. SVM is also available in the scikit-learn library and we follow the same structure for using it(Import library, object creation, fitting model and prediction).

Now, let us have a look at a real-life problem statement and dataset to understand how to apply SVM for classification

Problem Statement

Dream Housing Finance company deals in all home loans. They have a presence across all urban, semi-urban and rural areas. A customer first applies for a home loan, after that the company validates the customer's eligibility for a loan.

Company wants to automate the loan eligibility process (real-time) based on customer details provided while filling an online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers' segments, those are eligible for loan amount so that they can specifically target these customers. Here they have provided a partial data set.

Use the coding window below to predict the loan eligibility on the test set. **Try changing the hyperparameters for the linear SVM to improve the accuracy.**

Support Vector Machine(SVM) code in R

The e1071 package in R is used to create Support Vector Machines with ease. It has helper functions as well as code for the Naive Bayes Classifier. The creation of a support vector machine in R and Python follow similar approaches, let's take a look now at the following code:

```
#Import Library
require(e1071) #Contains the SVM
Train <- read.csv(file.choose())
Test <- read.csv(file.choose())
# there are various options associated with SVM training; like
changing kernel, gamma and C value.

# create model
model <-
svm(Target~Predictor1+Predictor2+Predictor3,data=Train,kernel='linear',gamma=0.2,cost=100)

#Predict Output
preds <- predict(model,Test)
table(preds)
```

How to tune Parameters of SVM?

Tuning the parameters' values for machine learning algorithms effectively improves model performance. Let's look at the list of parameters available with SVM.

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200,
class_weight=None, verbose=False, max_iter=-1, random_state=None)
```

I am going to discuss about some important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

kernel: We have already discussed about it. Here, we have various options available with kernel like, “linear”, “rbf”, “poly” and others (default value is “rbf”). Here “rbf” and “poly” are useful for non-linear hyper-plane. Let’s look at the example, where we’ve used linear kernel on two feature of iris data set to classify their class.

Support Vector Machine(SVM) code in Python

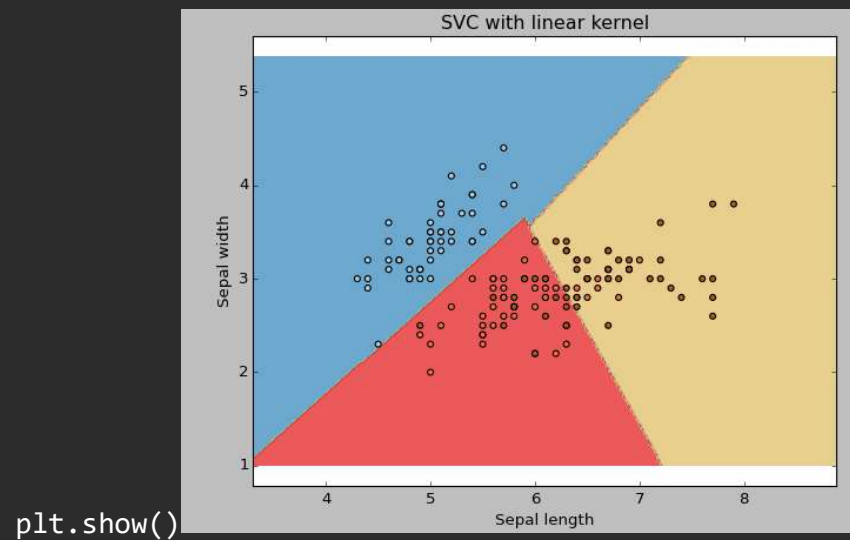
Example: Have a linear SVM kernel

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
# avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1, gamma=0).fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
```

```

Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')

```



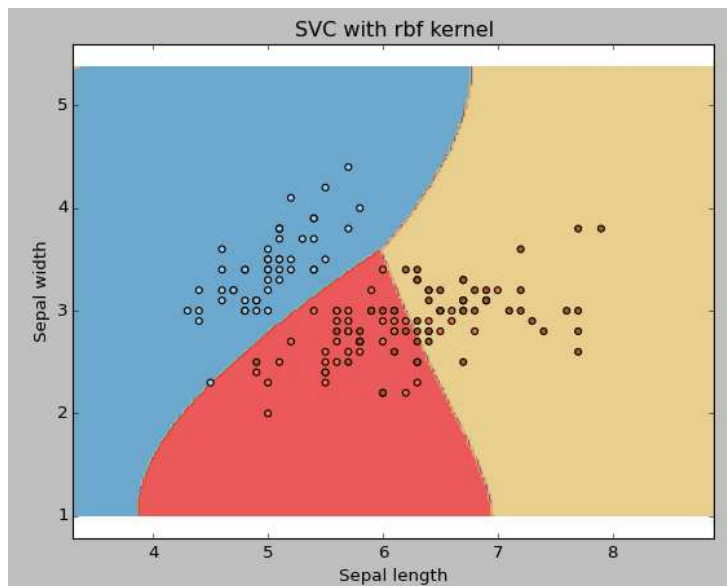
Example: Use SVM rbf kernel

Change the kernel type to rbf in below line and look at the impact.

```

svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)

```

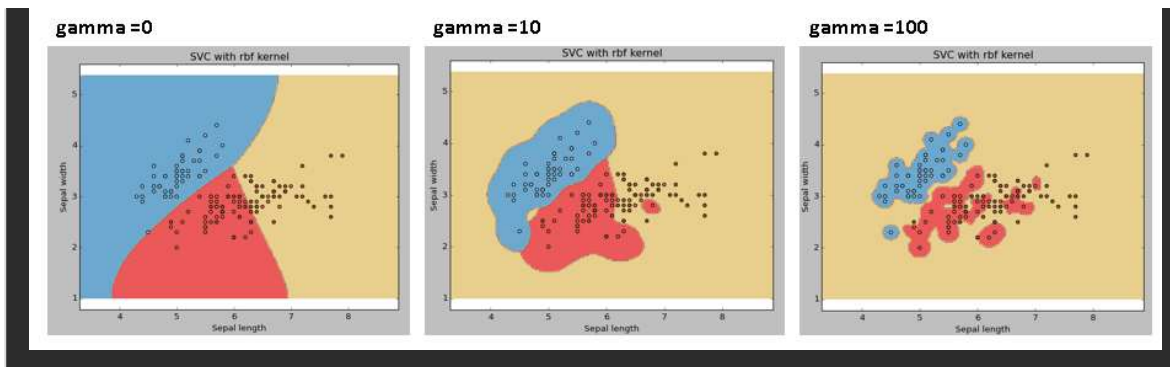


I would suggest you go for linear SVM kernel if you have a large number of features (>1000) because it is more likely that the data is linearly separable in high dimensional space. Also, you can use RBF but do not forget to cross-validate for its parameters to avoid over-fitting.

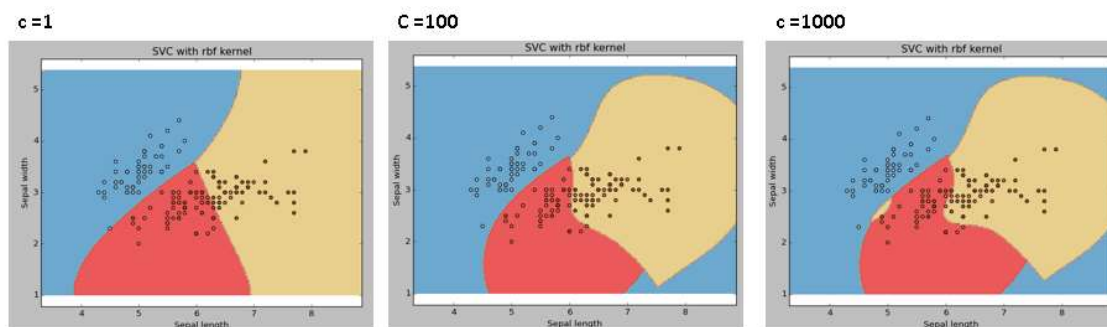
gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

Example: Let's difference if we have gamma different gamma values like 0, 10 or 100.

```
svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)
```



C: Penalty parameter C of the error term. It also controls the trade-off between smooth decision boundaries and classifying the training points correctly.



We should always look at the cross-validation score to have effective combination of these parameters and avoid over-fitting.

In R, SVMs can be tuned in a similar fashion as they are in Python.

Mentioned below are the respective parameters for e1071 package:

- The kernel parameter can be tuned to take "Linear", "Poly", "rbf" etc.
- The gamma value can be tuned by setting the "Gamma" parameter.
- The C value in Python is tuned by the "Cost" parameter in R.

Pros and Cons associated with SVM

- **Pros:**
 - It works really well with a clear margin of separation

- It is effective in high dimensional spaces.
- It is effective in cases where the 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 included in the related SVC method of Python scikit-learn library.