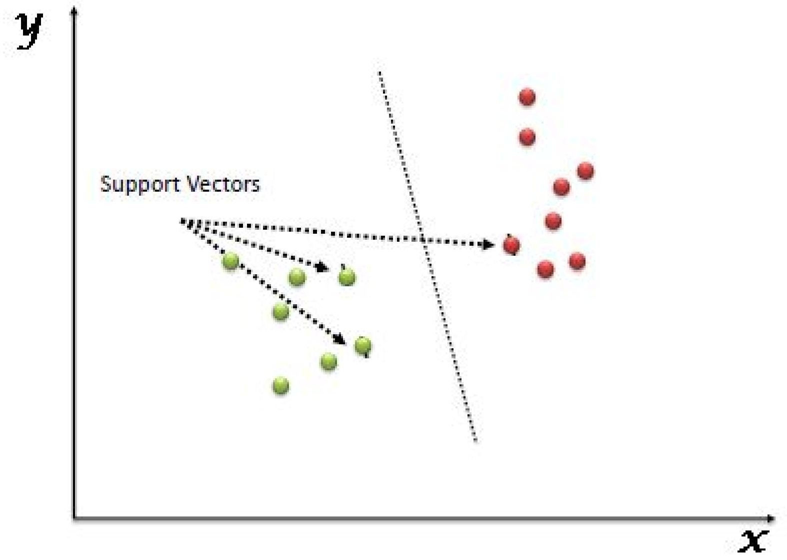
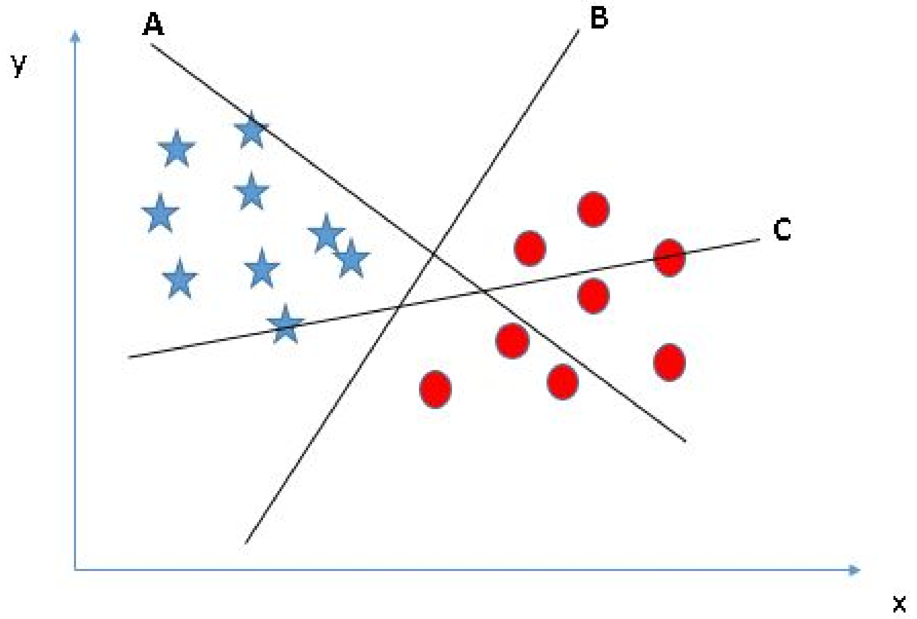
Support Vector Machines

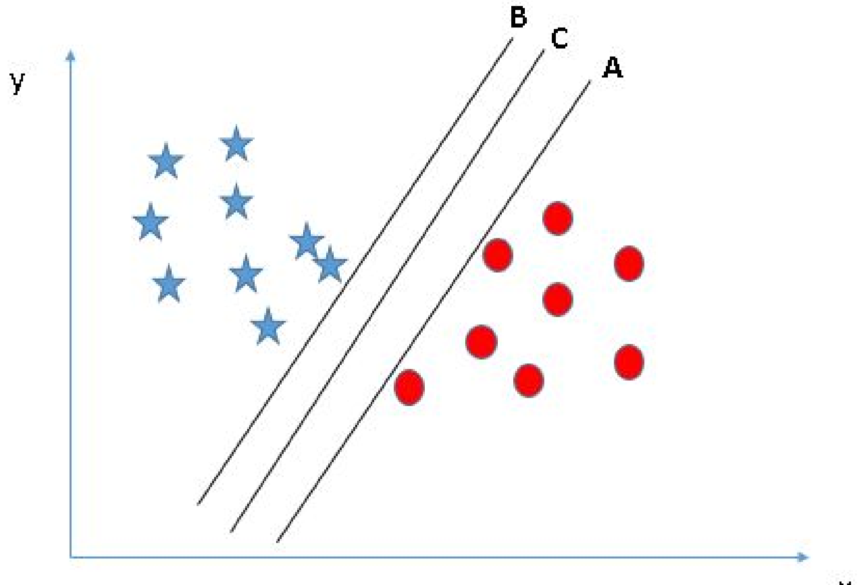
“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiate the two classes very well (look at the below snapshot).



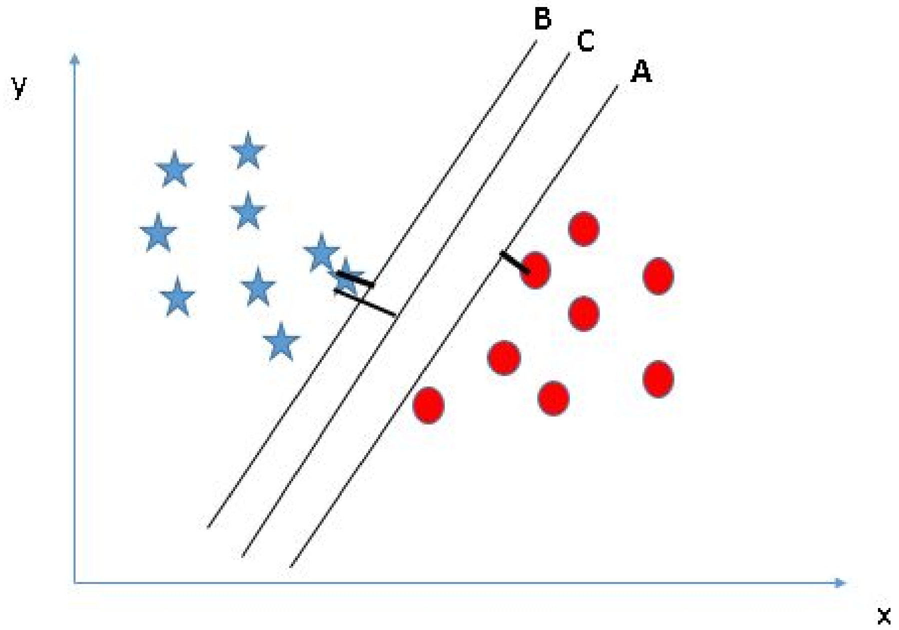
- **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.
- You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In 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:

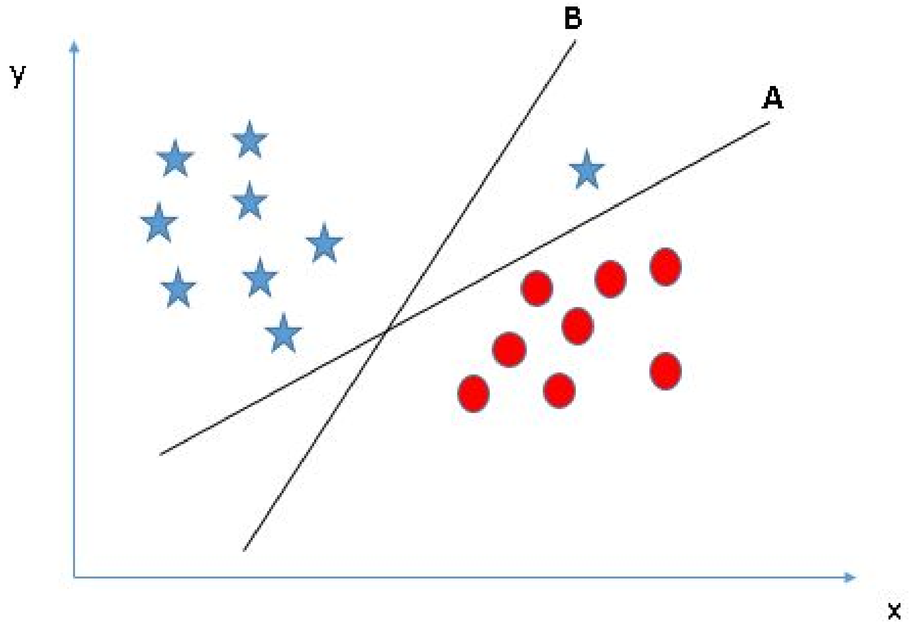


Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

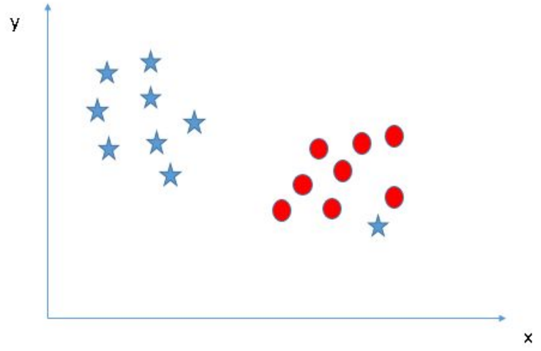


- **Identify the right hyper-plane (Scenario-3):**Hint: Use the rules as discussed in previous section to identify the right hyper-plane

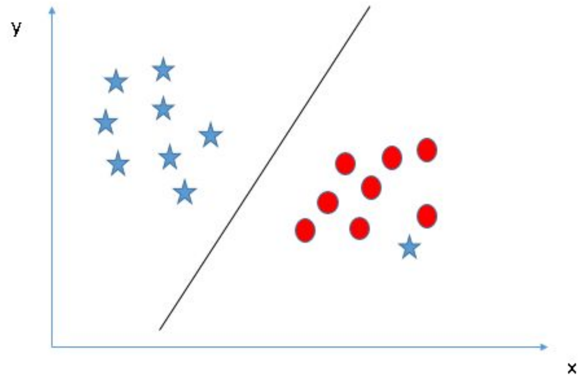
Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**



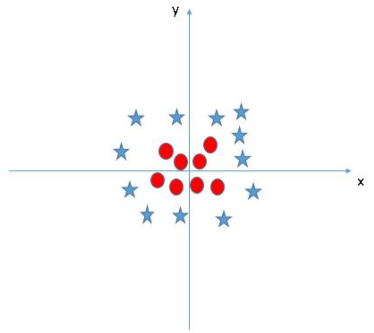
Can we classify two classes (Scenario-4)?: Below, I am unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.



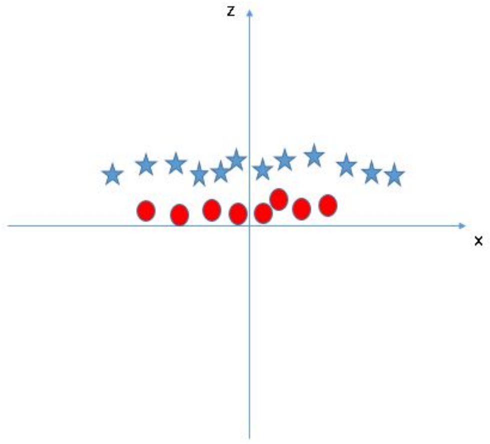
As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.



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=xˆ2+yˆ2. Now, let’s plot the data points on axis x and z:

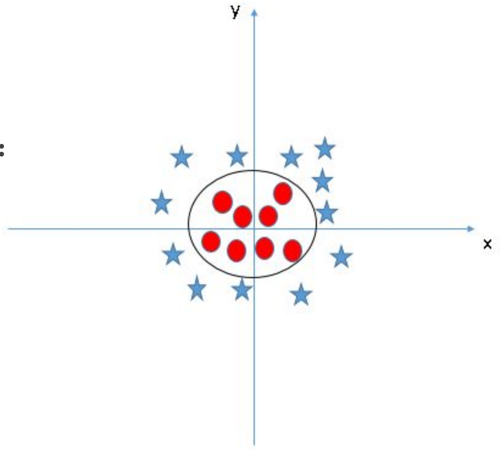


In above plot, points to consider are:

- 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 SVM, 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, SVM has a technique called the **kernel trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find 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:

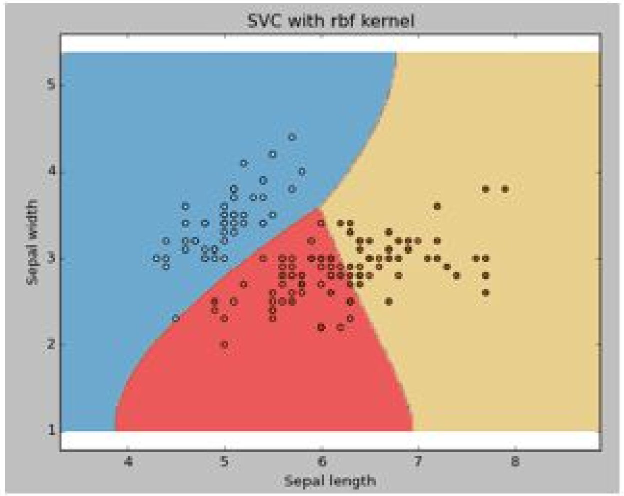


gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

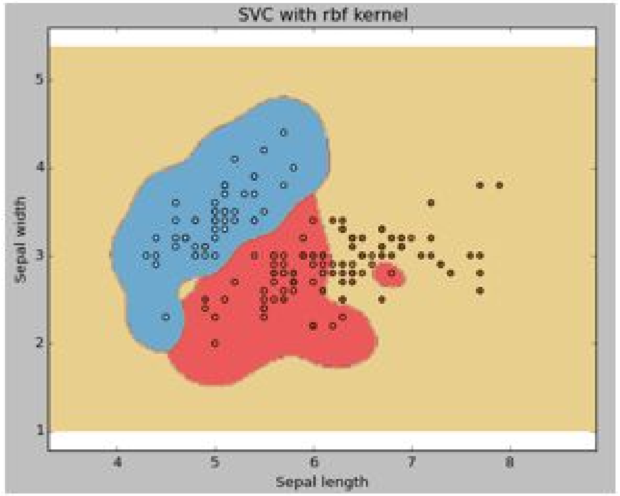
Example: Let's difference if we have gamma different gamma values like 0, 10 or 100.

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

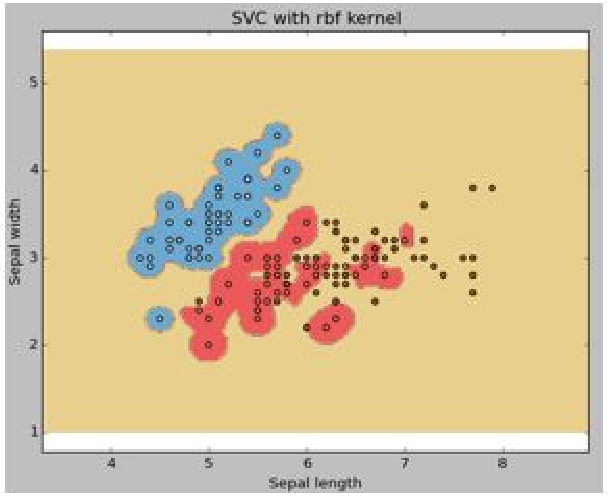
gamma =0



gamma =10

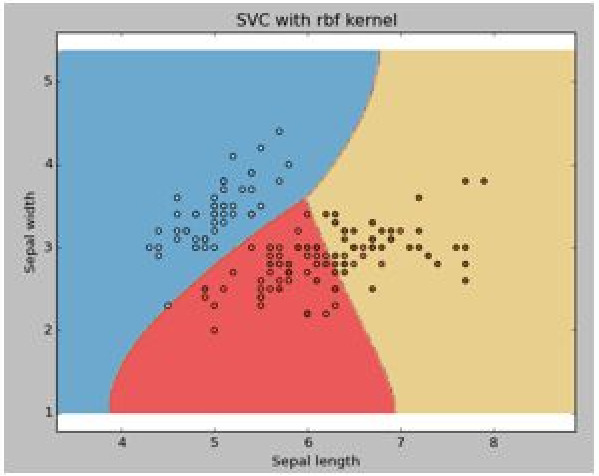


gamma =100

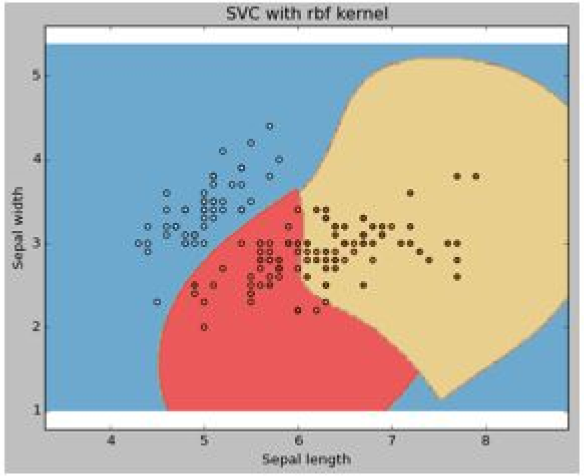


C: Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.

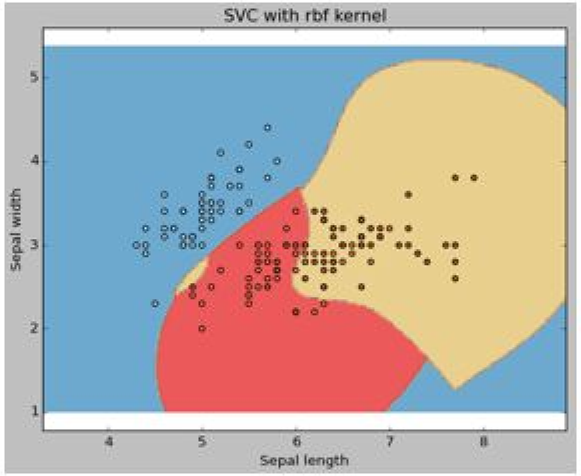
c =1



C =100



c =1000



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.

```r
library("e1071")
```

Using Iris data

```r
head(iris,5)
```

```
##   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
```

Attach the Data

```r
attach(iris)
```

Divide Iris data to x (containt the all features) and y only the classes

```r
x <- subset(iris, select=-Species)
y <- Species
```

Create SVM Model and show summary

```r
svm_model <- svm(Species ~ ., data=iris)
summary(svm_model)
```

Or you can use command like this

Create SVM Model and show summary

```
svm_model1 <- svm(x,y)
summary(svm_model1)
```

Run Prediction and you can measuring the execution time in R

```
pred <- predict(svm_model1,x)
system.time(pred <- predict(svm_model1,x))
```

```
##    user  system elapsed
##       0       0       0
```

See the confusion matrix result of prediction, using command table to compare the result of SVM prediction and the class data in y variable.

```
table(pred,y)
```

```
##             y
## pred         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         2
##   virginica       0          2        48
```

Tuning SVM to find the best cost and gamma ..

```r
svm_tune <- tune(svm, train.x=x, train.y=y,
              kernel="radial", ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)))

print(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.05333
```

After you find the best cost and gamma, you can create svm model again and try to run again

```
svm_model_after_tune <- svm(Species ~ ., data=iris, kernel="radial", cost=1, gamma=0.5)
summary(svm_model_after_tune)
```

Run Prediction again with new model

```
pred <- predict(svm_model_after_tune,x)
system.time(predict(svm_model_after_tune,x))
```

```
##    user  system elapsed
##       0       0       0
```

See the confusion matrix result of prediction, using command table to compare the result of SVM prediction and the class data in y variable.

```
table(pred,y)
```

```
##             y
## pred         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         2
##   virginica       0          2        48
```