‘j

Contents

[What is Support Vector Machine? 1](#_Toc520413133)

[what the heck is hyperplane? 2](#_Toc520413134)

[How is SVM’s hyperplane different from linear classifiers? 2](#_Toc520413135)

[How does it work? 2](#_Toc520413136)

[Various Scenarios 4](#_Toc520413137)

[Tuning parameters: Kernel,Regularization, Gamma and Margin 8](#_Toc520413138)

[Kernel 8](#_Toc520413139)

[Regularization(C,Gamma) 9](#_Toc520413140)

[How to implement SVM in Python and R? 11](#_Toc520413141)

[Describe SVM To a 5 year old 12](#_Toc520413142)

[Pros and Cons of SVM 12](#_Toc520413143)

# <https://www.hackerearth.com/blog/machine-learning/simple-tutorial-svm-parameter-tuning-python-r/>

https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

# What is Support Vector Machine?

“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.

Given a set of points of two types in N dimensional place SVM generates a (N−1) dimensional hyperplane to separate those points into two groups.

 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 hyper-plane that differentiate the two classes very well (look at the below snapshot).

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_1.png)

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

You can look at [definition of support vectors](https://www.analyticsvidhya.com/blog/2014/10/support-vector-machine-simplified/) and a few examples of its working here.

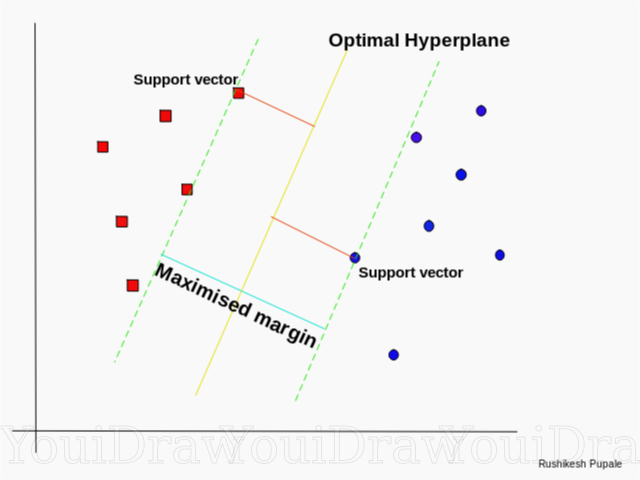
# what the heck is hyperplane?

The answer is “a line in more that 3 dimensions” ( in 1-D it’s called a point, in 2-D it’s called a line, in 3-D it’s called a plane, more than 3 - Hyperplane).

# How is SVM’s hyperplane different from linear classifiers?

Motivation: ***Maximize margin***: we want to find the classifier whose decision boundary is furthest away from any data point.

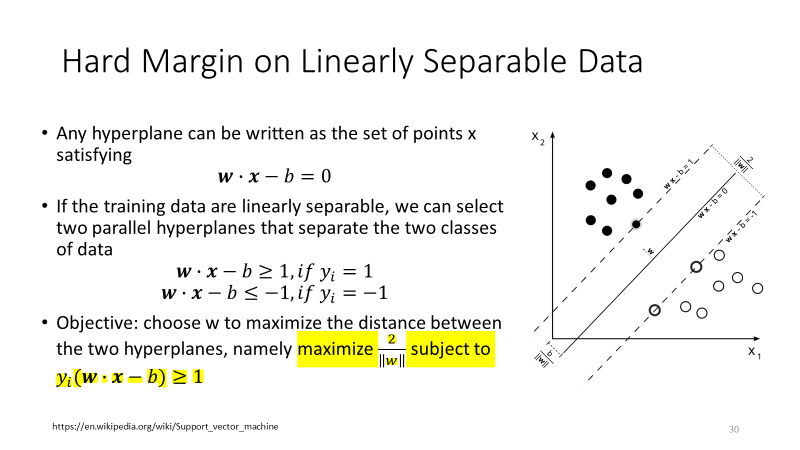
We can express the separating hyper-plane in terms of the data points that are closest to the boundary. And these points are called **support vectors.** **Now, we compute the distance between the line and the support vectors.** This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.

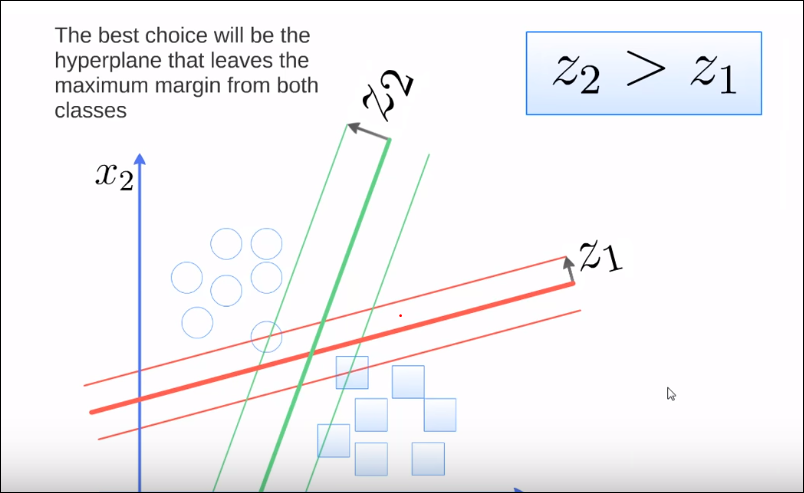


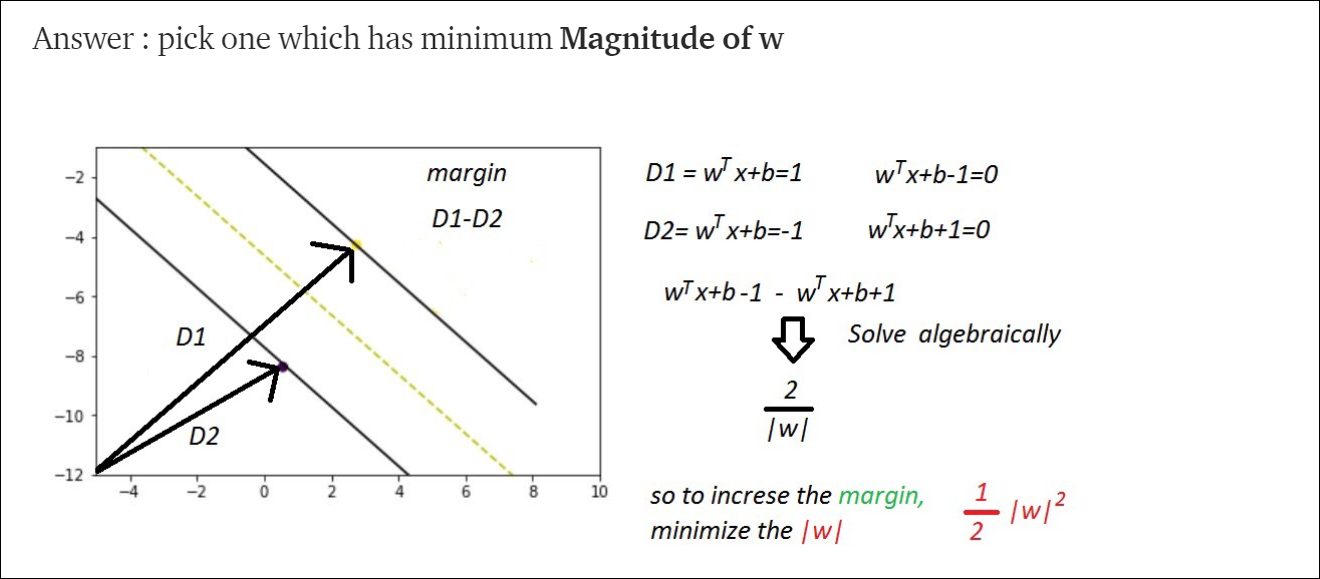
We would like to learn the weights that maximize the margin. So we have the hyperplane!

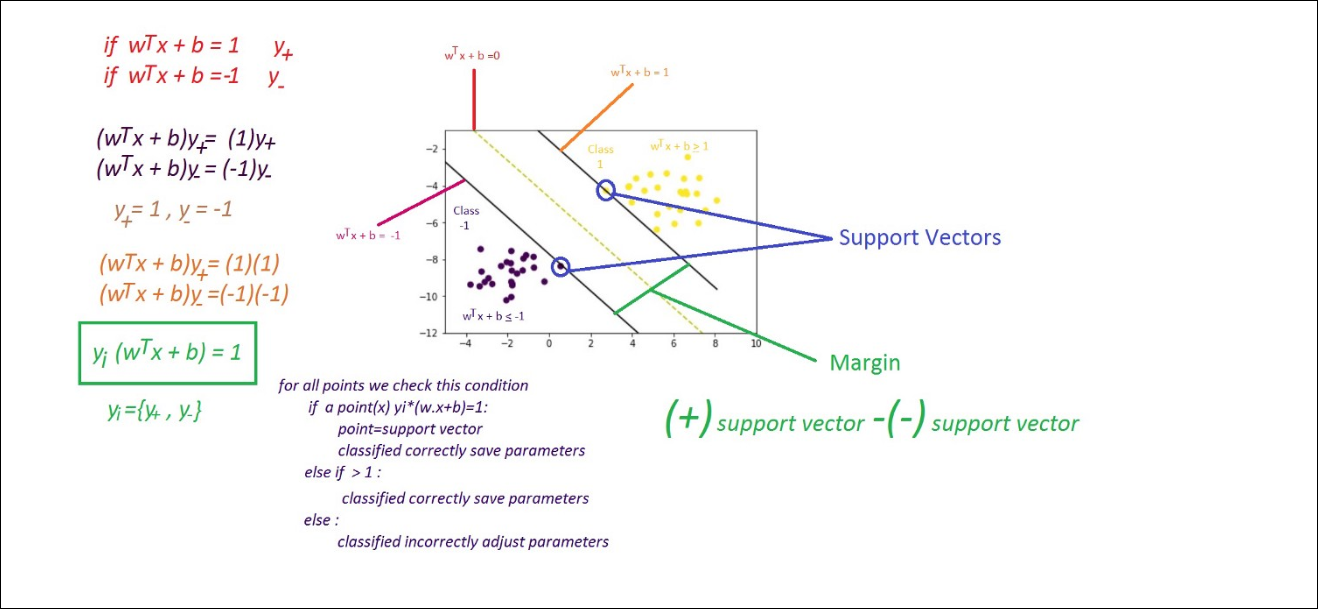
*Margin is the distance between the left hyperplane and right hyperplane.*

# 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!

# Various Scenarios

* **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:[[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_4.png)](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_4.png)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

**[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_5.png)**

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.  **[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_61.png)**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.  
  **[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_71.png)**

**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.**[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_8.png)**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:  
[[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_9.png)](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_9.png)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](https://en.wikipedia.org/wiki/Kernel_method" \t "_blank)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.

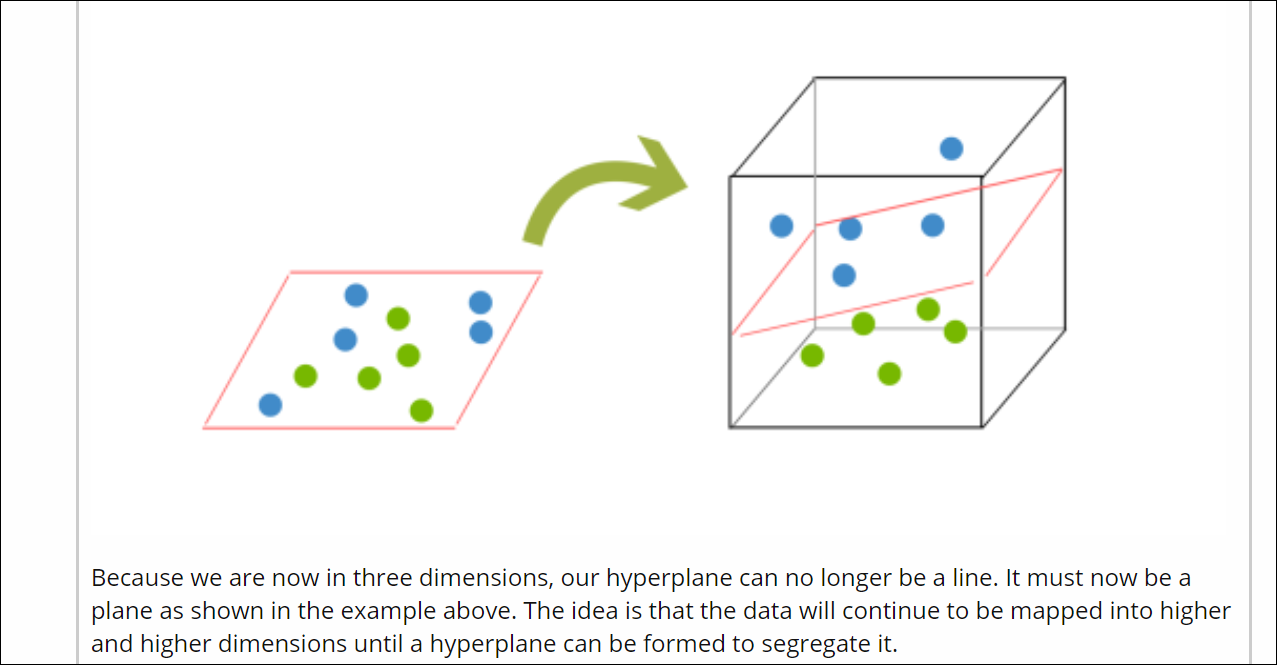
When we look at the hyper-plane in original input space it looks like a circle:  
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_10.png)

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

# Tuning parameters: Kernel,Regularization, Gamma and Margin

## **Kernel**

In order to classify a dataset like the one above it’s necessary to move away from a 2d view of the data to a 3d view. Explaining this is easiest with another simplified example. Imagine that our two sets of colored balls above are sitting on a sheet and this sheet is lifted suddenly, launching the balls into the air. While the balls are up in the air, you use the sheet to separate them. This ‘lifting’ of the balls represents the mapping of data into a higher dimension. This is known as kernelling. You can read more on Kerneling .



Support Vector Machines tend to find a linear decision boundary between points of two classes based on the maximum-margin principle, where the objective is to find a set of points which lie on two sides of the plane at a distance of at least unity. These points are the support vectors and the plane midway is the separating hyperplane which is always a linear plane.

Now, in practice, the dataset may not be linearly separable. Take the common example of a two-input XOR gate, with inputs x1 and x2 and output y. They are related as

x1    x2    y

0    0    0

0    1    1

1    0    1

1    1    0

Now if you plot these points in two dimensions with x1 and x2 as the features and y as the label, you can see that it is not possible to find a linear separating hyperplane that would separate the points of the two classes in this space of two dimensions.

Now, let us say we introduce a third dimension x3, which is computed as x3=(x1-x2)^2. The data projected in this high dimensional space will be

x1    x2    x3    y

0    0    0    0

0    1    1    1

1    0    1    1

1    1    0    0

Now if you visualize the data in this 3-dimensional space as shown below, you can see that it is linearly separable by a hyperplane.

I have not shown the hyperplane in the figure but it is easy to visualize several linear planes that can separate the above dataset. This is what kernels allow us to do. We can implicitly map the data to a higher dimensional space where it is linearly separable, and solve the SVM formulation in that space to obtain a linear decision boundary.

## 

# Tunig Parameter / Regularization(C,Gamma)

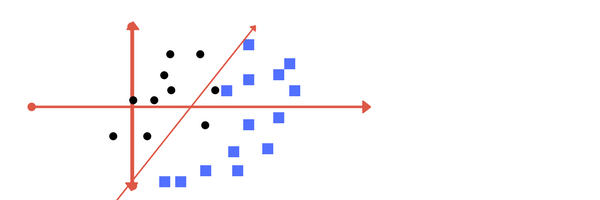
## C

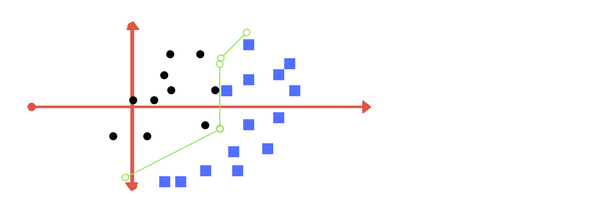
It controls the trade off between smooth decision boundary and classifying training points correctly. A large value of c means you will get more training points correctly.

The Regularization parameter (often termed as C parameter in python’s sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.

For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

The images below (same as image 1 and image 2 in section 2) are example of two different regularization parameter. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.





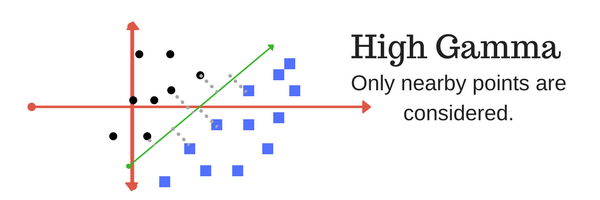
Left: low regularization value, right: high regularization value

## Gamma

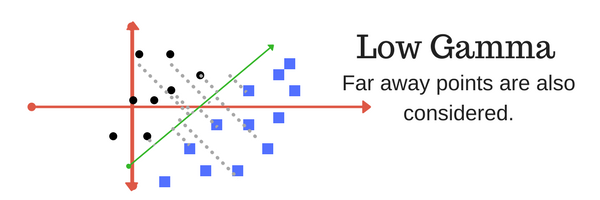
It defines how far the influence of a single training example reaches. If it has a low value it means that every point has a far reach and conversely high value of gamma means that every point has close reach.

If gamma has a very high value, then the decision boundary is just going to be dependent upon the points that are very close to the line which effectively results in ignoring some of the points that are very far from the decision boundary. This is because the closer points get more weight and it results in a wiggly curve as shown in previous graph.On the other hand, if the gamma value is low even the far away points get considerable weight and we get a more linear curve.

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible seperation line are considered in calculation for the seperation line. Where as high gamma means the points close to plausible line are considered in calculation.



High Gamma



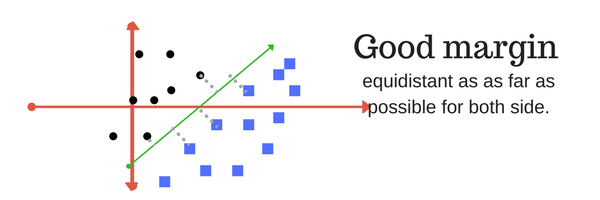
Low Gamma

## Margin

And finally last but very importrant characteristic of SVM classifier. SVM to core tries to achieve a good margin.

*A margin is a separation of line to the closest class points.*

A *good margin* is one where this separation is larger for both the classes. Images below gives to visual example of good and bad margin. A good margin allows the points to be in their respective classes without crossing to other class.



# 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 scikit-learn library and follow the same structure (Import library, object creation, fitting model and prediction). Let’s look at the below code:

#Import Library

from sklearn import svm

#Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset

# Create SVM classification object

model = svm.svc(kernel='linear', c=1, gamma=1)

# there is various option associated with it, like changing kernel, gamma and C value. Will discuss more # about it in next section.Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x\_test)

# Describe SVM To a 5 year old

# How it is different from Logistic regression

Few major things of SVM that are conceptually different from a logistic regression —

**Part#1**: Loss function

**Part#2**: Maximum margin classification — At a very fundamental level, in SVM, a line L1 is said to be a better classifier than line L2, if the “margin” of L1 is larger i.e., L1 is farther from both classes.

**Part#3**: Feature transformation using Kernel trick

Lets go over one by one —

**Loss function:**First, lets start with Loss function —

Lets take an example of a simple binary classification task. Then for the given input features “X” and target “y”, the goal of the SVM algorithm is to predict a value ( ‘predicted y’) close to the target (‘actual y’) for each observation. To do this —

We are interested in a equation that could calculate ‘predicted y’. This equation depends on some weighted values of input X. It can be written as :

***‘predicted y’ = f (weighted values of X).***

Lets denote our weights as *w*.

We can start the classification by drawing a random decision boundary ( aka., this is same as predicting some random values for ‘predicted y’) — and this is exactly what is happening when we initialize the weights in the above equation with random values.

Identify the cost/loss function. This job of the loss function is to quantify the error between the ‘predicted y’ and the ‘actual y’. To put this in a simple way, this defines the amount by which you want to penalize the mis-classified observations. So intuitively, the farther the ‘predicted y’ from the ‘actual y’, more should be the penalty and vice-versa. Given the amount of total error ( = sum of losses of for each observation) we made in a particular iteration, we try to reduce this error by adjusting the weights for the next iteration. This continues till we can no longer can minimize the Total error / cost**.**This total error / cost function is as below:

***Total error / cost = Sum of all losses for each observation in that iteration.***

4. The final weights which we arrive at, at the end of all iterations forms our final model used for predicting on unseen data.

So, for SVM, a loss function called as ‘hinge loss’ is used — refer to the below plot (*from wikipedia)*.

**Plot of hinge loss on y-axis and ‘predicted y’ on x-axis.**

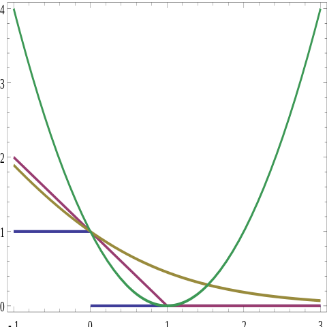
Note that from the above plot (blue line), it can be seen that the loss equals 0 , when

the ‘predicted *y’* ≥1 and

‘actual *y’*and the *‘*predicted *y’* have the same sign (meaning ‘predicted *y’*predicts the right class)

But when ‘actual *y’ , ‘*predicted *y’*have opposite sign, the hinge loss increases linearly with *y* (one-sided error).

SVM uses hinge loss where as logistic regression using logistic loss function for optimizing the cost function and arriving at the weights. The way the hinge loss is different from logistic loss can be understood from the plot below (*from wikipedia — Purple is the hinge loss, Yellow is the logistic loss function*).



Plot of various loss functions — Purple is the hinge loss function. Yellow is the logistic loss function.

Note that the yellow line gradually curves downwards unlike purple line where the loss becomes 0 for values ‘predicted y’ ≥1. By looking at the plots above, this nature of curves brings out few major differences between logistic loss and hinge loss —

Note that the logistic loss diverges faster than hinge loss. So, in general, it will be more sensitive to outliers — why? Because, assuming there is an outlier in our data, the logistic loss (due to its diverging nature) will be very high compared to the hinge loss for that outlier. This means greater adjustments to our weights.

Note that the logistic loss does not go to zero even if the point is classified sufficiently confidently — The horizontal axis being the confidence of ‘predicted y’ value, if we take a value like ‘1.5’ on x-axis, then the corresponding logistic loss (yellow line) still shows some loss (close to 0.2 from the above plot and hence still not very confident of our prediction), whereas the hinge loss is ‘0’ ( which means there is no loss and we are more confident of our prediction). This nature of logistic loss might lead to minor degradation in accuracy.

Logistic regression has a more probabilistic interpretation.

Given this understanding of the hinge loss function for a SVM, lets add a regularization term (L2 norm) to the cost. The intuition behind the regularization term is that we increase the cost penalty if the values for the weights are high. So while trying to minimize the cost, we not only adjust the weights, we also try to minimize the value of the weights and thereby reduce over fitting to the training data and make the model less sensitive to outliers.

So with the added regularization term, the total cost function finally looks like:

***Total cost = ||w²||/2 + C\*(Sum of all losses for each observation)***

where ‘C’ is the hyper-parameter that controls the amount of regularization.

# Pros and Cons of SVM

**Pros**

Accuracy

Works well on smaller cleaner datasets

It can be more efficient because it uses a subset of training points

Firstly it has a regularisation parameter, which makes the user think about avoiding over-fitting.

Secondly it uses the kernel trick, so you can build in expert knowledge about the problem via engineering the kernel.

Thirdly an SVM is defined by a convex optimisation problem (no local minima) for which there are efficient methods (e.g. SMO).

Lastly, it is an approximation to a bound on the test error rate, and there is a substantial body of theory behind it which suggests it should be a good idea.

**Cons**

Isn’t suited to larger datasets as the training time with SVMs can be high

Less effective on noisier datasets with overlapping classes( more sensitive to outliers because of Hinge loss function )