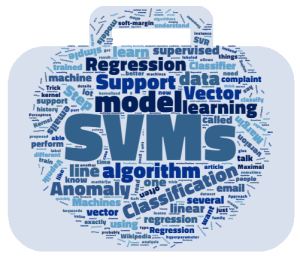
VMs - An overview of Support Vector Machines

[17 Replies](https://www.svm-tutorial.com/2017/02/svms-overview-support-vector-machines/#comments)



Today we are going to talk about SVMs in general.

I recently received an email from a reader of [my serie of articles about the math behind SVM](https://www.svm-tutorial.com/2014/11/svm-understanding-math-part-1/):

*I felt I got deviated a lot on Math part and its derivations and assumptions and finally got confused what exactly SVM is ? And when to use it and how it helps ?*

Here is my attempt to clarify things.

**What exactly is SVM ?**

**SVM is a *supervised* learning model**

It means you need a dataset which has been **labeled**.

**Exemple**: I have a business and I receive a lot of emails from customers every day. Some of these emails are complaints and should be answered very quickly. I would like a way to identify them quickly so that I answer these email in priority.

Approach 1: I can create a label in gmail using keywords, for instance "urgent", "complaint", "help"

The drawback of this method is that I need to think of all potential keywords that some angry users might use, and I will probably miss some of them. Over time, my keyword list will probably become very messy and it will be hard to maintain.

Approach 2: I can use a supervised machine learning algorithm.

Step 1: I need a lot of emails, the more the better.  
Step 2: I will read the title of each email and classify it by saying "it is a complaint" or "it is not a complaint".  It put a **label**on each email.  
Step 3: I will **train** a model on this dataset  
Step 4: I will assess the quality of the prediction (using cross validation)  
Step 5: I will use this model to **predict** if an email is a complaint or not.

In this case, if I have trained the model with a lot of emails then it will perform well. SVM is just one among many models you can use to learn from this data and make predictions.

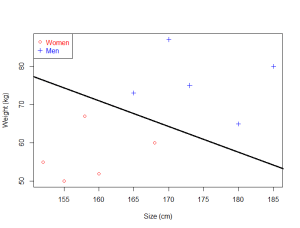
Note that the **crucial** part is Step 2. If you give SVM **unlabeled** emails, then it can do nothing.

**SVM learns*a linear model***

Now we saw in our previous example that at the Step 3 a supervised learning algorithm such as SVM is **trained** with the labeled data. But what is it trained for? It is trained to learn something.  
What does it learn?

In the case of SVM, it learns a **linear model**.

What is a linear model? In simple words: it is a line (in complicated words it is a hyperplane).  
If your data is very simple and only has two dimensions, then the SVM will learn a line which will be able to separate the data.



*The SVM is able to find a line which separates the data*

If it is just a line, why do we talk about a linear**model**?  
Because you cannot **learn** a line.

So instead of that:

* 1) We suppose that the data we want to classify can be separated by a line
* 2) We know that a line can be represented by the equation y=wx+by=wx+b (this is our model)
* 3) We know that there is an infinity of possible lines obtained by changing the value of wwand bb
* 4) We use an algorithm to determine which are the values of ww and bb giving the "best" line separating the data.

SVM is one of these algorithms.

**Algorithm or model?**

At the start of the article I said SVM is a supervised learning **model**, and now I say it is an **algorithm**.  What's wrong? The term algorithm is often loosely used. For instance, you will sometime read that SVM is a supervised learning algorithm. This is not true if you consider that an algorithm is a set of actions to perform to obtain a specific result. [Sequential minimal optimization](https://en.wikipedia.org/wiki/Sequential_minimal_optimization) is the most used algorithm to train SVM, but you can train an SVM with another algorithm like [Coordinate descent](https://en.wikipedia.org/wiki/Coordinate_descent). However, most people are not interested in details like this, so we simplify and say that we use the SVM "algorithm" (without saying in details which one we use).

**SVM or SVMs?**

Sometime, you will see people talk about SVM, and sometime about SVMs.

As often Wikipedia is quite good at stating things clearly:

*In machine learning, support vector machines (****SVMs****)****are supervised learning models with associated learning algorithms****that analyze data used for classification and regression analysis.*[*(Wikipedia)*](https://en.wikipedia.org/wiki/Support_vector_machine)

So, we now discover that there are several models, which belongs to the SVM family.

**SVMs - Support Vector Machines**

Wikipedia tells us that SVMs can be used to do two things: classification or regression.

* **SVM** is used for classification
* **SVR** (Support Vector Regression) is used for regression

So it makes sense to say that there are several Support Vector Machines. However, this is not the end of the story !

**Classification**

In 1957, a simple linear model called the [Perceptron](https://en.wikipedia.org/wiki/Perceptron)was invented by [Frank Rosenblatt](https://en.wikipedia.org/wiki/Frank_Rosenblatt) to do classification (which is in fact one of the building block of simple neural networks also called [Multilayer Perceptron](https://en.wikipedia.org/wiki/Multilayer_perceptron)).

A few years later,  [Vapnik](https://en.wikipedia.org/wiki/Vladimir_Vapnik" \t "_blank)and [Chervonenkis](https://en.wikipedia.org/wiki/Alexey_Chervonenkis), proposed another model called the "Maximal Margin Classifier", the SVM was born.

Then, in 1992, Vapnik et al. had the idea to apply what is called the [Kernel Trick](http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html), which allow to use the SVM to classify linearly nonseparable data.

Eventually, in 1995, Cortes and Vapnik introduced the Soft Margin Classifier which allows us to accept some misclassifications when using a SVM.

So just when we talk about classification there is already**four different Support Vector Machines**:

1. The original one : the Maximal Margin Classifier,
2. The kernelized version using the Kernel Trick,
3. The soft-margin version,
4. The soft-margin kernelized version (which combine 1, 2 and 3)

And this is of course the last one which is used most of the time. That is why SVMs can be tricky to understand at first, because they are made of several pieces which came with time.

That is why when you use a programming language you are often asked to specify which kernel you want to use (because of the kernel trick), and which value of the hyperparameter C you want to use (because it controls the effect of the soft-margin).

**Regression**

In 1996, Vapnik et al. proposed a version of SVM to perform regression instead of classification. It is called Support Vector Regression (SVR). Like the classification SVM, this model includes the C hyperparameter and the kernel trick.

I wrote a[simple article, explaining how to use SVR in R](https://www.svm-tutorial.com/2014/10/support-vector-regression-r/).

If you wish to learn more about SVR, you can read this [good tutorial by Smola and Schölkopft](http://www.svms.org/regression/SmSc98.pdf).

**Summary of the history**

* Maximal Margin Classifier (1963 or 1979)
* Kernel Trick (1992)
* Soft Margin Classifier (1995)
* Support Vector Regression (1996)

If you want to know more, you can learn this [very detailed overview of the history](http://www.svms.org/history.html).

**Other type of Support Vector Machines**

Because SVMs have been very successful at classification, people started to think about using the same logic for other type of problems, or to create derivation. As a result there exists now several different and interesting methods in the SVM family:

* [Structured support vector machine](https://en.wikipedia.org/wiki/Structured_support_vector_machine) which is able to predict [structured objects](https://en.wikipedia.org/wiki/Structured_prediction)
* [Least square support vector machine](https://en.wikipedia.org/wiki/Least_squares_support_vector_machine) used for classification and regression
* [Support vector clustering](http://www.scholarpedia.org/article/Support_vector_clustering) used to perform [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis)
* [Transductive Support Vector Machine](https://en.wikipedia.org/wiki/Transduction_(machine_learning)) used for [semi-supervised learning](https://en.wikipedia.org/wiki/Semi-supervised_learning)
* [Ranking SVM](https://en.wikipedia.org/wiki/Ranking_SVM) used to sort results
* [One class support vector machine](http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/) used for [anomaly detection](https://en.wikipedia.org/wiki/Anomaly_detection)

**Conclusion**

We have learned that it is normal to have some difficulty to understand what SVM is exactly. This is because there are several Support Vector Machines used for different purposes. As often, history allows us to have a better vision of how the SVM we know today has been built.

I hope this article give you a broader view of the SVM panorama, and will allow you to understand these machines better.

If you wish to learn more about how SVM work for classification, you can start reading the math series:

**SVM - Understanding the math**

[**Part 1: What is the goal of the Support Vector Machine (SVM)?**](https://www.svm-tutorial.com/2014/11/svm-understanding-math-part-1/)  
[Part 2: How to compute the margin?](https://www.svm-tutorial.com/2014/11/svm-understanding-math-part-2/)[Part 3: How to find the optimal hyperplane?](https://www.svm-tutorial.com/2015/06/svm-understanding-math-part-3/)[Part 4: Unconstrained minimization](https://www.svm-tutorial.com/2016/09/unconstrained-minimization/)  
[Part 5: Convex functions](https://www.svm-tutorial.com/2016/09/convex-functions/)  
[Part 6: Duality and Lagrange multipliers](https://www.svm-tutorial.com/2016/09/duality-lagrange-multipliers/)



[Alexandre KOWALCZYK](https://www.svm-tutorial.com/author/alexandrekowgmail-com/)

*I am passionate about machine learning and Support Vector Machine. I like to explain things simply to share my knowledge with people from around the world. If you wish you can add me to linkedin, I like to connect with my readers.*

This entry was posted in [SVM Tutorial](https://www.svm-tutorial.com/svm-tutorial/) and tagged [SVM](https://www.svm-tutorial.com/tag/svm/), [SVMs](https://www.svm-tutorial.com/tag/svms/), [SVR](https://www.svm-tutorial.com/tag/svr/) on [February 26, 2017](https://www.svm-tutorial.com/2017/02/svms-overview-support-vector-machines/).

**Post navigation**

[← SVM - Understanding the math - Duality and Lagrange multipliers](https://www.svm-tutorial.com/2016/09/duality-lagrange-multipliers/)[Support Vector Machines Succinctly released →](https://www.svm-tutorial.com/2017/10/support-vector-machines-succinctly-released/)

17 thoughts on “SVMs - An overview of Support Vector Machines”

In this guide I want to introduce you to an extremely powerful machine learning technique known as the **Support Vector Machine** (SVM). It is one of the best "out of the box" supervised classification techniques. As such, it is an important tool for both the quantitative trading researcher and data scientist.

I feel it is important for a quant researcher or data scientist to be comfortable with both the theoretical aspects and practical usage of the techniques in their toolkit. Hence this article will form the first part in a series of articles that discuss support vector machines. This article specifically will cover the theory of **maximal margin classifiers**, **support vector classifiers** and **support vector machines**. Subsequent articles will make use of the Python [scikit-learn](http://scikit-learn.org/stable/) library to demonstrate some examples of the aforementioned theoretical techniques on actual data.

## Motivation for Support Vector Machines

The problem to be solved in this article is one of **supervised binary classification**. That is, we wish to categorise new unseen objects into two separate groups based on their properties and a set of known examples, which are already categorised. A good example of such a system is classifying a set of new documents into positive or negative sentiment groups, based on other documents which have already been classified as positive or negative. Similarly, we could classify new emails into spam or non-spam, based on a large corpus of documents that have already been marked as spam or non-spam by humans. SVMs are highly applicable to such situations.

A Support Vector Machine models the situation by creating a feature space, which is a finite-dimensional [vector space](https://en.wikipedia.org/wiki/Vector_space), each dimension of which represents a "feature" of a particular object. In the context of spam or document classification, each "feature" is the prevalence or importance of a particular word.

The goal of the SVM is to train a model that assigns new unseen objects into a particular category. It achieves this by creating a linear partition of the feature space into two categories. Based on the features in the new unseen objects (e.g. documents/emails), it places an object "above" or "below" the separation plane, leading to a categorisation (e.g. spam or non-spam). This makes it an example of a non-probabilistic linear classifier. It is non-probabilistic, because the features in the new objects fully determine its location in feature space and there is no stochastic element involved.

However, much of the benefit of SVMs comes from the fact that they are not restricted to being linear classifiers. Utilising a technique known as the **kernel trick** they can become much more flexible by introducing various types of non-linear decision boundaries.

Formally, in mathematical language, SVMs construct linear separating hyperplanes in high-dimensional vector spaces. Data points are viewed as (x⃗ ,y)(x→,y) tuples, x⃗ =(x1,…,xp)x→=(x1,…,xp) where the xjxj are the feature values and yy is the classification (usually given as +1+1 or −1−1). Optimal classification occurs when such hyperplanes provide maximal distance to the nearest training data points. Intuitively, this makes sense, as if the points are well separated, the classification between two groups is much clearer.

However, if in a feature space some of the sets are not linearly separable (i.e. they overlap!), then it is necessary to perform a [mapping](https://en.wikipedia.org/wiki/Map_%28mathematics%29) of the original feature space to a higher-dimensional space, in which the separation between the groups is clear, or at least clearer. However, this has the consequence of making the separation boundary in the original space potentially non-linear.

In this article we will proceed by considering the advantages and disadvantages of SVMs as a classification technique, then defining the concept of an **optimal linear separating hyperplane**, which motivates a simple type of linear classifier known as a maximal margin classifier (MMC). We will then show that maximal margin classifiers are not often applicable to many "real world" situations and as such need modification, in the form of a support vector classifier (SVC). We will then relax the restriction of linearity and consider non-linear classifiers, namely support vector machines, which use **kernel functions** to improve computational efficiency.

### Advantages and Disadvantages of SVMs

As a classification technique, the SVM has many advantages, many of which are due to its computational efficiency on large datasets. The Scikit-Learn team have summarised the main advantages and disadvantages [here](http://scikit-learn.org/stable/modules/svm.html) but I have repeated and elaborated on them for completeness:

#### **Advantages**

* **High-Dimensionality** - The SVM is an effective tool in high-dimensional spaces, which is particularly applicable to document classification and sentiment analysis where the dimensionality can be extremely large (≥106≥106).
* **Memory Efficiency** - Since only a subset of the training points are used in the actual decision process of assigning new members, only these points need to be stored in memory (and calculated upon) when making decisions.
* **Versatility** - Class separation is often highly non-linear. The ability to apply new kernels allows substantial flexibility for the decision boundaries, leading to greater classification performance.

#### **Disadvantages**

* p>np>n - In situations where the number of features for each object (pp) exceeds the number of training data samples (nn), SVMs can perform poorly. This can be seen intuitively, as if the high-dimensional feature space is much larger than the samples, then there are less effective support vectors on which to support the optimal linear hyperplanes, leading to poorer classification performance as new unseen samples are added.
* **Non-Probabilistic** - Since the classifier works by placing objects above and below a classifying hyperplane, there is no direct probabilistic interpretation for group membership. However, one potential metric to determine "effectiveness" of the classification is how far from the decision boundary the new point is.

Now that we've outlined the advantages and disadvantages, we're going to discuss the geometric objects and mathematical entities that will ultimately allow us to define the SVMs and how they work.

There are some fantastic references (both links and textbooks) that derive much of the mathematical detail of how SVMs work. In the following derivation I didn't want to "reinvent the wheel" too much, especially with regards notation and pedagogy, so I've formulated the following treatment based on the references provided at the end of the article, making strong use of [James et al (2013)](https://www.quantstart.com/articles/Support-Vector-Machines-A-Guide-for-Beginners#ref-isl), [Hastie et al (2009)](https://www.quantstart.com/articles/Support-Vector-Machines-A-Guide-for-Beginners#ref-esl)and the [Wikibooks article on SVMs](https://en.wikibooks.org/wiki/Support_Vector_Machines). I have made changes to the notation where appropriate and have adjusted the narrative to suit individuals interested in quantitative finance.

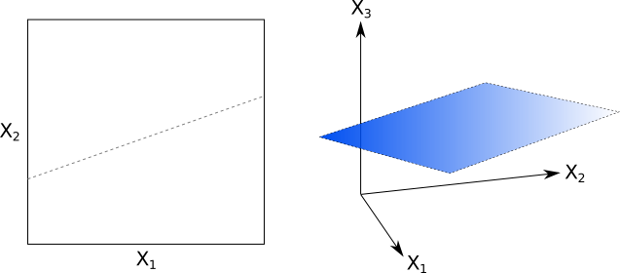
## Linear Separating Hyperplanes

The linear separating hyperplane is the key geometric entity that is at the heart of the SVM. Informally, if we have a high-dimensional feature space, then the linear hyperplane is an object one dimension lower than this space that divides the feature space into two regions.

This linear separating plane need not pass through the origin of our feature space, i.e. it does not need to include the zero vector as an entity within the plane. Such hyperplanes are known as **affine**.

If we consider a real-valued pp-dimensional feature space, known mathematically as RpRp, then our linear separating hyperplane is an affine p−1p−1 dimensional space embedded within it.

For the case of p=2p=2 this hyperplane is simply a one-dimensional straight line, which lives in the larger two-dimensional plane, whereas for p=3p=3 the hyerplane is a two-dimensional plane that lives in the larger three-dimensional feature space (see Fig 1 and Fig 2):

  
**Figs 1 and 2: One- and two-dimensional hyperplanes**

If we consider an element of our pp-dimensional feature space, i.e. x⃗ =(x1,...,xp)∈Rpx→=(x1,...,xp)∈Rp, then we can mathematically define an affine hyperplane by the following equation:

b0+b1x1+...+bpxp=0b0+b1x1+...+bpxp=0

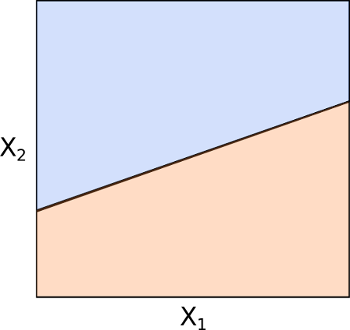
b0≠0b0≠0 gives us an affine plane (i.e. it does not pass through the origin). We can use a more succinct notation for this equation by introducing the summation sign:

b0+∑j=1pbjxj=0b0+∑j=1pbjxj=0

Notice however that this is nothing more than a multi-dimensional [dot product](https://en.wikipedia.org/wiki/Dot_product) (or, more generally, an [inner product](https://en.wikipedia.org/wiki/Inner_product_space)), and as such can be written even more succinctly as:

b⃗ ⋅x⃗ +b0=0b→⋅x→+b0=0

If an element x⃗ ∈Rpx→∈Rp satisfies this relation then it lives on the p−1p−1-dimensional hyperplane. This hyperplane splits the pp-dimensional feature space into two classification regions (see Fig 3):

  
**Fig 3: Separation of**pp**-dimensional space by a hyperplane**

Elements x⃗ x→ above the plane satisfy:

b⃗ ⋅x⃗ +b0>0b→⋅x→+b0>0

While those below it satisfy:

b⃗ ⋅x⃗ +b0<0b→⋅x→+b0<0

The key point here is that it is possible for us to determine which side of the plane any element x⃗ x→ will fall on by calculating the [sign](https://en.wikipedia.org/wiki/Sign_%28mathematics%29) of the expression b⃗ ⋅x⃗ +b0b→⋅x→+b0. This concept will form the basis of a supervised classification technique.

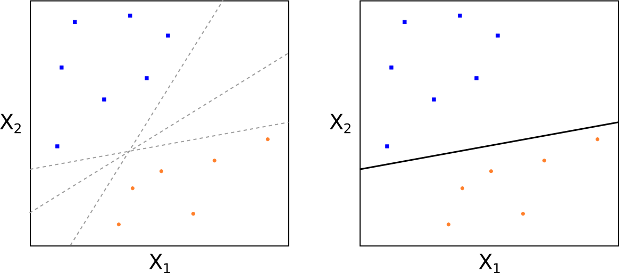
## Classification

Continuing with our example of email spam filtering, we can think of our classification problem (say) as being provided with a thousand emails (n=1000n=1000), each of which is marked spam (+1+1) or non-spam (−1−1). In addition, each email has an associated set of keywords (i.e. separating the words on spacing) that provide features. Hence if we take the set of all possible keywords from all of the emails (and remove duplicates), we will be left with pp keywords in total.

If we translate this into a mathematical problem, the standard setup for a supervised classification procedure is to consider a set of nn training observations, x⃗ ix→i, each of which is a pp-dimensional vector of features. Each training observation has an associated class label, yi∈{−1,1}yi∈{−1,1}. Hence we can think of nn pairs of training observations (x⃗ i,yi)(x→i,yi) representing the features and class labels (keyword lists and spam/non-spam). In addition to the training observations we can provide test observations, x⃗ ∗=(x∗1,...,x∗p)x→∗=(x1∗,...,xp∗) that are later used to test the performance of the classifiers. In our spam example, these test observations would be new emails that have not yet been seen.

Our goal is to develop a classifier based on provided training observations that will correctly classify subsequent test observations using only their feature values. This translates into being able to classify an email as spam or non-spam solely based on the keywords contained within it.

We will initially suppose that it is possible, via a means yet to be determined, to construct a hyperplane that separates training data perfectly according to their class labels (see Figs 4 and 5). This would mean cleanly separating spam emails from non-spam emails solely by using specific keywords. The following diagram is only showing p=2p=2, while for keyword lists we may have p>106p>106. Hence Figs 4 and 5 are only representative of the problem.

  
**Fig 4: Multiple separating hyperplanes; Fig 5: Perfect separation of class data**

This translates into a mathematical separating property of:

b⃗ ⋅x⃗ i+b0>0,ifyi=1b→⋅x→i+b0>0,ifyi=1

and

b⃗ ⋅x⃗ i+b0<0,ifyi=−1b→⋅x→i+b0<0,ifyi=−1

This basically states that if each training observation is above or below the separating hyperplane, according to the geometric equation which defines the plane, then its associated class label will be +1+1or −1−1. Thus we have developed a simple classification process. We assign a test observation to a class depending upon which side of the hyperplane it is located on.

This can be formalised by considering the following function f(x⃗ )f(x→), with a test observation x⃗ ∗=(x∗1,...,x∗p)x→∗=(x1∗,...,xp∗):

f(x⃗ ∗)=b⃗ ⋅x⃗ ∗+b0f(x→∗)=b→⋅x→∗+b0

If f(x⃗ ∗)>0f(x→∗)>0 then y∗=+1y∗=+1, whereas if f(x⃗ ∗)<0f(x→∗)<0 then y∗=−1y∗=−1.

However, this tells us nothing about how we go about finding the bjbj components of b⃗ b→, as well as b0b0, which are crucial in helping us determine the equation of the hyperplane separating the two regions. The next section discusses an approach for carrying this out, as well as introducing the concept of the **maximal margin hyperplane** and a classifier built on it, known as the **maximal margin classifier**.

## Deriving the Classifier

At this stage it is worth pointing out that separating hyperplanes are not unique, since it is possible to slightly translate or rotate such a plane without touching any training observations (see Fig 4).

So, not only do we need to know how to construct such a plane, but we also need to determine the most optimal. This motivates the concept of the **maximal margin hyperplane** (MMH), which is the separating hyperplane that is farthest from any training observations, and is thus "optimal".

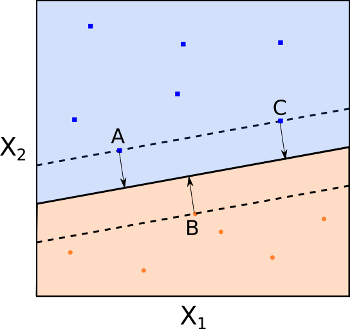
How do we find the maximal margin hyperplane? Firstly, we compute the perpendicular distance from each training observation x⃗ ix→i for a given separating hyperplane. The smallest perpendicular distance to a training observation from the hyperplane is known as the **margin**. The MMH is the separating hyperplane where the margin is the largest. This guarantees that it is the farthest minimum distance to a training observation.

The classification procedure is then just simply a case of determining which side a test observation falls on. This can be carried out using the above formula for f(x⃗ ∗)f(x→∗). Such a classifier is known as a **maximimal margin classifier** (MMC). Note however that finding the particular values that lead to the MMH is purely based on the training observations. That is, we still need to be aware of how the MMC performs on the test observations. We are implicitly making the assumption that a large margin in the training observations will provide a large margin on the test observations, but this may not be the case.

As always, we must be careful to avoid overfitting when the number of feature dimensions is large (e.g. in Natural Language Processing applications such as email spam classification). Overfitting here means that the MMH is a very good fit for the training data but can perform quite poorly when exposed to testing data. I discuss this issue in depth in the [article on the bias-variance trade-off](https://www.quantstart.com/articles/The-Bias-Variance-Tradeoff-in-Statistical-Machine-Learning-The-Regression-Setting).

To reiterate, our goal now becomes finding an algorithm that can produce the bjbj values, which will fix the geometry of the hyperplane and hence allow determination of f(x⃗ ∗)f(x→∗) for any test observation.

If we consider Fig 6, we can see that the MMH is the mid-line of the widest "block" that we can insert between the two classes such that they are perfectly separated.

  
**Fig 6: Maximal margin hyperplane with support vectors (A, B and C)**

One of the key features of the MMC (and subsequently SVC and SVM) is that the location of the MMH only depends on the **support vectors**, which are the training observations that lie directly on the margin (but not hyperplane) boundary (see points A, B and C in Fig 6). This means that the location of the MMH is NOT dependent upon any other training observations.

Thus it can be immediately seen that a potential drawback of the MMC is that its MMH (and thus its classification performance) can be extremely sensitive to the support vector locations. However, it is also partially this feature that makes the SVM an attractive computational tool, as we only need to store the support vectors in memory once it has been "trained" (i.e. the bjbj values are fixed).

## Constructing the Maximal Margin Classifier

I feel it is instructive to fully outline the optimisation problem that needs to be solved in order to create the MMH (and thus the MMC itself). While I will outline the constraints of the optimisation problem, the algorithmic solution to this problem is beyond the scope of the article. Thankfully these optimisation routines are implemented in scikit-learn (actually, via the [LIBSVM library](https://www.csie.ntu.edu.tw/~cjlin/libsvm/)). If you wish to read more about the solution to these algorithmic problems, take a look at [*Hastie et al (2009)*](https://www.quantstart.com/articles/Support-Vector-Machines-A-Guide-for-Beginners#ref-esl) and the [*Scikit-Learn page on Support Vector Machines*](http://scikit-learn.org/stable/modules/svm.html).

The procedure for determining a maximal margin hyperplane for a maximal margin classifier is as follows. Given nn training observations x⃗ 1,...,x⃗ n∈Rpx→1,...,x→n∈Rp and nn class labels y1,...,yn∈{−1,1}y1,...,yn∈{−1,1}, the MMH is the solution to the following optimisation procedure:

Maximise M∈RM∈R, by varying b1,...,bpb1,...,bp such that:

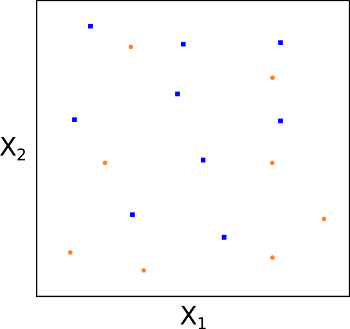
∑j=1pb2j=1∑j=1pbj2=1

and

yi(b⃗ ⋅x⃗ +b0)≥M,∀i=1,...,nyi(b→⋅x→+b0)≥M,∀i=1,...,n

Despite the complex looking constraints, they actually state that each observation must be on the correct side of the hyperplane and at least a distance MM from it. Since the goal of the procedure is to maximise MM, this is precisely the condition we need to create the MMC!

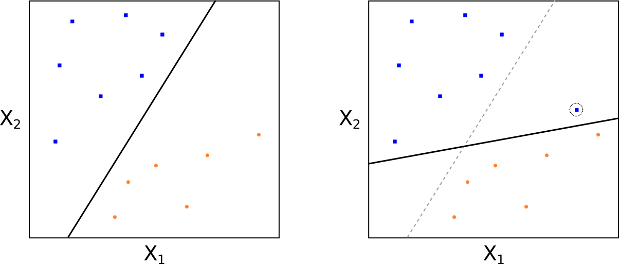
Clearly, the case of perfect separability is an ideal one. Most "real world" datasets will not have such perfect separability via a linear hyperplane (see Fig 7). However, if there is no separability then we are unable to construct a MMC by the optimisation procedure above. So, how do we create a form of separating hyperplane?

  
**Fig 7: No possibility of a true separating hyperplane**

Essentially we have to relax the requirement that a separating hyperplane will perfectly separate every training observation on the correct side of the line (i.e. guarantee that it is associated with its true class label), using what is called a **soft margin**. This motivates the concept of a **support vector classifier**(SVC).

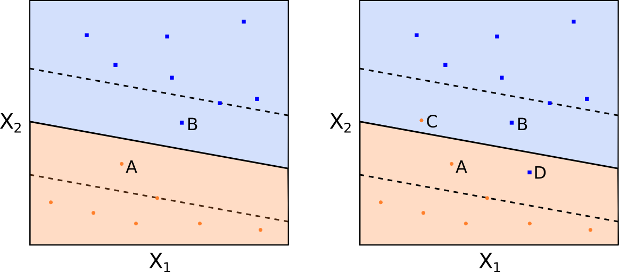
## Support Vector Classifiers

As we alluded to above, one of the problems with MMC is that they can be extremely sensitive to the addition of new training observations. Consider Figs 8 and 9. In Fig 8 it can be seen that there exists a MMH perfectly separating the two classes. However, in Fig 9 if we add one point to the +1+1 class we see that the location of the MMH changes substantially. Hence in this situation the MMH has clearly been over-fit:

  
**Figs 8 and 9: Addition of a single point dramatically changes the MMH line**

As we mentioned above also, we could consider a classifier based on a separating hyperplane that doesn't perfectly separate the two classes, but does have a greater robustness to the addition of newinvididual observations and has a better classification on most of the training observations. This comes at the expense of some misclassification of a few training observations.

This is how a support vector classifier or soft margin classifier works. A SVC allows some observations to be on the incorrect side of the margin (or hyperplane), hence it provides a "soft" separation. The following figures 10 and 11 demonstrate observations being on the wrong side of the margin and the wrong side of the hyperplane respectively:

  
**Figs 10 and 11: Observations on the wrong side of the margin and hyperplane, respectively**

As before, an observation is classified depending upon which side of the separating hyperplane it lies on, but some points may be misclassified.

It is instructive to see how the optimisation procedure differs from that described above for the MMC. We need to introduce new parameters, namely nn ϵiϵi values (known as the slack values) and a parameter CC, known as the budget. We wish to maximise MM, across b1,...,bp,ϵ1,..,ϵnb1,...,bp,ϵ1,..,ϵn such that:

∑j=1pb2j=1∑j=1pbj2=1

and

yi(b⃗ ⋅x⃗ +b0)≥M(1−ϵi),∀i=1,...,nyi(b→⋅x→+b0)≥M(1−ϵi),∀i=1,...,n

and

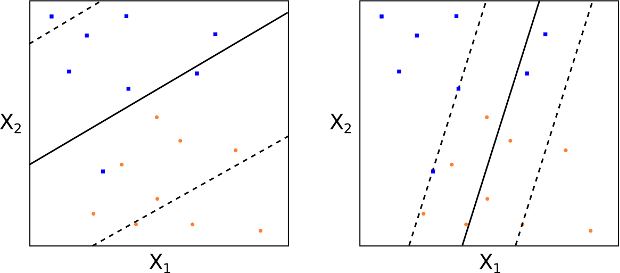
ϵi≥0,∑i=1nϵi≤Cϵi≥0,∑i=1nϵi≤C

Where CC, the budget, is a non-negative "tuning" parameter. MM still represents the margin and the slack variables ϵiϵi allow the individual observations to be on the wrong side of the margin or hyperplane.

In essence the ϵiϵi tell us where the iith observation is located relative to the margin and hyperplane. For ϵi=0ϵi=0 it states that the xixi training observation is on the correct side of the margin. For ϵi>0ϵi>0 we have that xixi is on the wrong side of the margin, while for ϵi>1ϵi>1 we have that xixi is on the wrong side of the hyperplane.

CC collectively controls how much the individual ϵiϵi can be modified to violate the margin. C=0C=0implies that ϵi=0,∀iϵi=0,∀i and thus no violation of the margin is possible, in which case (for separable classes) we have the MMC situation.

For C>0C>0 it means that no more than CC observations can violate the hyperplane. As CC increases the margin will widen. See Fig 12 and 13 for two differing values of CC:

  
**Figs 12 and 13: Different values of the tuning parameter**CC

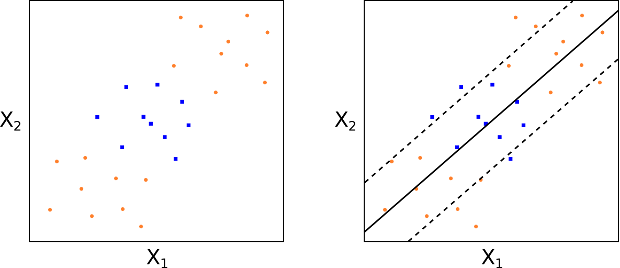
How do we choose CC in practice? Generally this is done via [cross-validation](https://www.quantstart.com/articles/Using-Cross-Validation-to-Optimise-a-Machine-Learning-Method-The-Regression-Setting). In essence CC is the parameter that governs the [bias-variance trade-off](https://www.quantstart.com/articles/The-Bias-Variance-Tradeoff-in-Statistical-Machine-Learning-The-Regression-Setting) for the SVC. A small value of CC means a low bias, high variance situation. A large value of CC means a high bias, low variance situation.

As before, to classify a new test observation x∗x∗ we simply calculate the sign of f(x⃗ ∗)=b⃗ ⋅x⃗ ∗+b0f(x→∗)=b→⋅x→∗+b0.

This is all well and good for classes that are linearly (or nearly linearly) separated. However, what about separation boundaries that are non-linear? How do we deal with those situations? This is where we can extend the concept of support vector classifiers to support vector machines.

## Support Vector Machines

The motivation behind the extension of a SVC is to allow non-linear decision boundaries. This is the domain of the Support Vector Machine (SVM). Consider the following Figs 14 and 15. In such a situation a purely linear SVC will have extremely poor performance, simply because the data has no clear linear separation:

  
**Figs 14 and 15: No clear linear separation between classes and thus poor SVC performance**

Hence SVCs can be useless in highly non-linear class boundary problems.

In order to motivate how an SVM works, we can consider a standard "trick" in linear regression, when considering non-linear situations. In particular a set of pp features x1,...,xpx1,...,xp can be transformed, say, into a set of 2p2p features x1,x21,...,xp,x2px1,x12,...,xp,xp2. This allows us to apply a linear technique to a set of non-linear features.

While the decision boundary is linear in the new 2p2p-dimensional feature space it is non-linear in the original pp-dimensional space. We end up with a decision boundary given by q(x⃗ )=0q(x→)=0 where qq is a quadratic polynomial function of the original features and hence is a non-linear solution.

This is clearly not restricted to quadratic polynomials. Higher dimensional polynomials, interaction terms and other functional forms, could all be considered. Although the drawback is that it dramatically increases the dimension of the feature space to the point that some algorithms can become untractable.

The major advantage of SVMs is that they allow a non-linear enlargening of the feature space, while still retaining a significant computational efficiency, using a process known as the ["kernel trick"](http://en.wikipedia.org/wiki/Kernel_method), which will be outlined below shortly.

So what are SVMs? In essence they are an extension of SVCs that results from enlargening the feature space through the use of functions known as **kernels**. In order to understand kernels, we need to briefly discuss some aspects of the solution to the SVC optimisation problem outlined above.

While calculating the solution to the SVC optimisation problem, the algorithm only needs to make use of **inner products** between the observations and not the observations themselves. Recall that an inner product is defined for two pp-dimensional vectors u,vu,v as:

⟨u⃗ ,v⃗ ⟩=∑j=1pujvj⟨u→,v→⟩=∑j=1pujvj

Hence for two observations an inner product is defined as:

⟨x⃗ i,x⃗ k⟩=∑j=1pxijxkj⟨x→i,x→k⟩=∑j=1pxijxkj

While we won't dwell on the details (since they are beyond the scope of this article), it is possible to show that a linear support vector classifier for a particular observation x⃗ x→ can be represented as a linear combination of inner products:

f(x⃗ )=b0+∑i=1nαi⟨x⃗ ,x⃗ i⟩f(x→)=b0+∑i=1nαi⟨x→,x→i⟩

With nn aiai coefficients, one for each of the training observations.

To estimate the b0b0 and aiai coefficients we only need to calculate (n2)=n(n−1)/2(n2)=n(n−1)/2 inner products between all pairs of training observations. In fact, we ONLY need to calculate the inner products for the subset of training observations that represent the support vectors. I will call this subset SS. This means that:

ai=0ifx⃗ i∉Sai=0ifx→i∉S

This means we can rewrite the representation formula as:

f(x)=b0+∑i∈Sai⟨x⃗ ,x⃗ i⟩f(x)=b0+∑i∈Sai⟨x→,x→i⟩

This turns out to be a major advantage for computational efficiency.

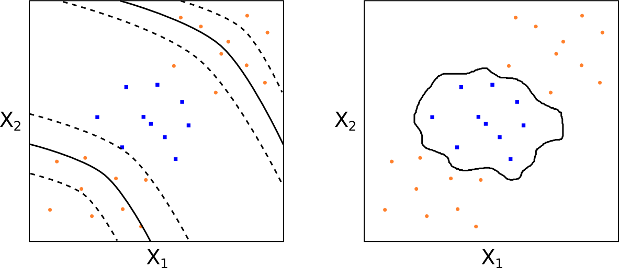
This now motivates the extension to SVMs. If we consider the inner product ⟨x⃗ i,x⃗ k⟩⟨x→i,x→k⟩ and replace it with a more general inner product "kernel" function K=K(x⃗ i,x⃗ k)K=K(x→i,x→k), we can modify the SVC representation to use non-linear kernel functions and thus modify how we calculate "similarity" between two observations. For instance, to recover the SVC we just take KK to be as follows:

K(x⃗ i,x⃗ k)=∑j=1pxijxkjK(x→i,x→k)=∑j=1pxijxkj

Since this kernel is linear in its features the SVC is known as the linear SVC. We can also consider polynomial kernels, of degree dd:

K(x⃗ i,x⃗ k)=(1+∑j=1pxijxkj)dK(x→i,x→k)=(1+∑j=1pxijxkj)d

This provides a significantly more flexible decision boundary and essentially amounts to fitting a SVC in a higher-dimensional feature space involving dd-degree polynomials of the features (see Fig 16).

  
**Fig 16: A**dd**-degree polynomial kernel; Fig 17: A radial kernel**

Hence, the definition of a support vector machine is a support vector classifier with a non-linear kernel function.

We can also consider the popular radial kernel (see Fig 17):

K(x⃗ i,x⃗ k)=exp(−γ∑j=1p(xij−xkj)2),γ>0K(x→i,x→k)=exp⁡(−γ∑j=1p(xij−xkj)2),γ>0

So how do radial kernels work? They are clearly quite different from polynomial kernels. Essentially if our test observation x⃗ ∗x→∗ is far from a training observation x⃗ ix→i in standard Euclidean distance then the sum ∑pj=1(x∗j−xij)2∑j=1p(xj∗−xij)2 will be large and thus K(x⃗ ∗,x⃗ i)K(x→∗,x→i) will be very small. Hence this particular training observation x⃗ ix→i will have almost no effect on where the test observation x⃗ ∗x→∗ is placed, via f(x⃗ ∗)f(x→∗).

Thus the radial kernel has extremely localised behaviour and only nearby training observations to x⃗ ∗x→∗will have an impact on its class label.

While this article has been very theoretical, the [next article on document classification using Scikit-Learn](https://www.quantstart.com/articles/Supervised-Learning-for-Document-Classification-with-Scikit-Learn) makes heavy use of SVMs in Python.

## Biblographic Notes

Originally, SVMs were invented by [Vapnik (1996)](https://www.quantstart.com/articles/Support-Vector-Machines-A-Guide-for-Beginners#ref-vapnik), while the current standard "soft margin" approach is due to [Cortes (1995)](https://www.quantstart.com/articles/Support-Vector-Machines-A-Guide-for-Beginners#ref-cortes). My treatment of the material follows, and is strongly influenced by, the excellent statistical machine learning texts of [James et al (2013)](https://www.quantstart.com/articles/Support-Vector-Machines-A-Guide-for-Beginners#ref-isl) and [Hastie et al (2009)](https://www.quantstart.com/articles/Support-Vector-Machines-A-Guide-for-Beginners#ref-esl).

## References

* Vapnik, V. (1996) The Nature of Statistical Learning Theory
* Cortes, C., Vapnik, V. (1995) "Support Vector Networks", Machine Learning **20** (3): 273
* James, G., Witten, D., Hastie, T., Tibshiranie, R. (2013) An Introduction to Statistical Learning
* Hastie, T., Tibshiranie, R., Friedman, J. (2009) The Elements of Statistical Learning
* Wikibooks (2016) Support Vector Machines ([Link](https://en.wikibooks.org/wiki/Support_Vector_Machines))
* Scikit-Learn (2016) Support Vector Machines ([Link](http://scikit-learn.org/stable/modules/svm.html))

able 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 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. 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.

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 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)**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.

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)

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 parameters value for machine learning algorithms effectively improves the 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.

**Example:**Have linear 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)

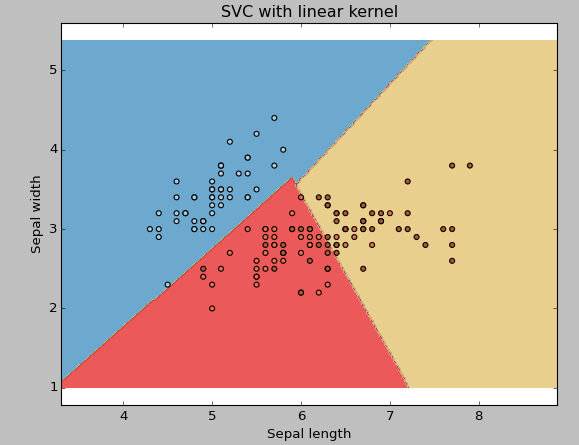
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

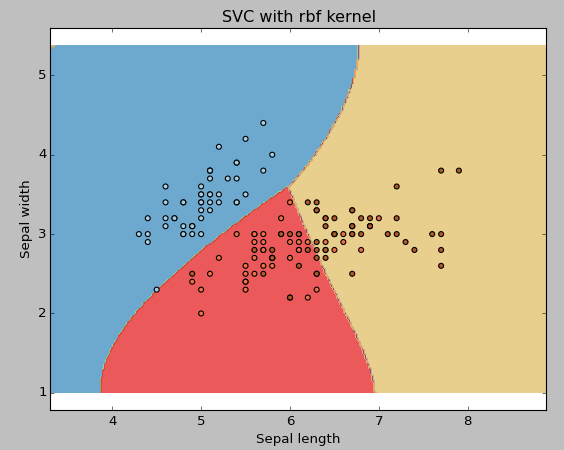
plt.title('SVC with linear kernel')

plt.show()[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_111.png)

**Example:**Have 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)

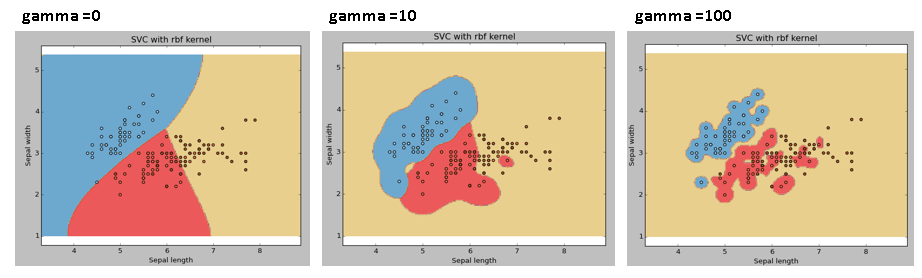
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_12.png)

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

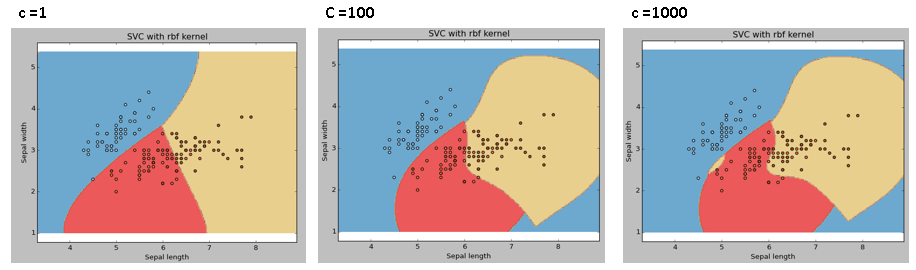
**gamma**: Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

**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)

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_15.png)

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

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_18.png)

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:

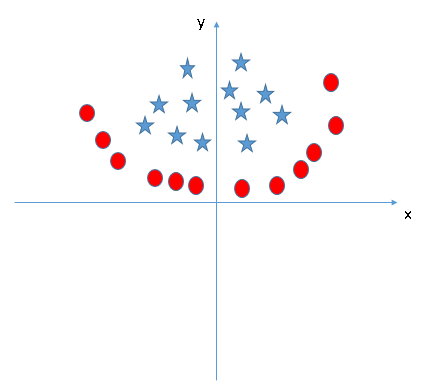
* The kernel parameter can be tuned to take “Linear”,”Poly”,”rbf” etc.
* The gamma value can be tuned by setting the “Gamma” parameter.
* The C value in Python is tuned by the “Cost” parameter in R.

Pros and Cons associated with SVM

* **Pros:**
  + It works really well with 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.

Practice Problem

Find right additional feature to have a hyper-plane for segregating the classes in below snapshot:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_19.png)

Answer the variable name in the comments section below. I’ll shall then reveal the answer.

End Notes

In this article, we looked at the machine learning algorithm, Support Vector Machine in detail.  I discussed its concept of working, process of implementation in python, the tricks to make the model efficient by tuning its parameters, Pros and Cons, and finally a problem to solve. I would suggest you to use SVM and analyse the power of this model by tuning the parameters. I also want to hear your experience with SVM, how have you tuned parameters to avoid over-fitting and reduce the training time?

Did you find this article helpful? Please share your opinions / thoughts in the comments section below.

If you like what you just read & want to continue your analytics learning, [subscribe to our emails](http://feedburner.google.com/fb/a/mailverify?uri=analyticsvidhya), [follow us on twitter](http://twitter.com/analyticsvidhya) or like our [facebook page](http://facebook.com/analyticsvidhya).

**Share this:**

* [89Click to share on LinkedIn (Opens in new window)89](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?share=linkedin&nb=1)
* [108Click to share on Facebook (Opens in new window)108](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?share=facebook&nb=1)
* [Click to share on Google+ (Opens in new window)](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?share=google-plus-1&nb=1)
* [Click to share on Twitter (Opens in new window)](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?share=twitter&nb=1)
* [Click to share on Pocket (Opens in new window)](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?share=pocket&nb=1)
* [Click to share on Reddit (Opens in new window)](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?share=reddit&nb=1)

**RELATED**