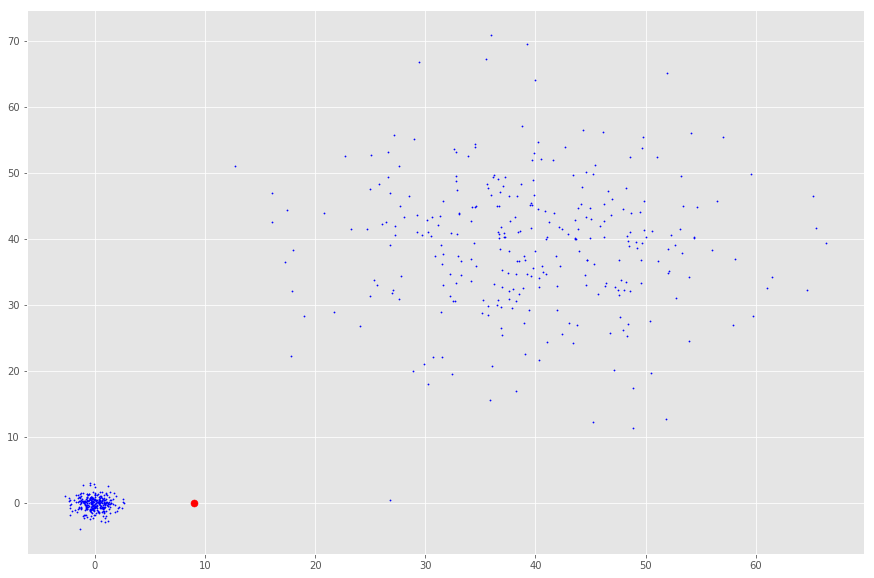
Local Outlier Factor (LOF) is a score that tells how likely a certain data point is an outlier/anomaly.

LOF ≈1 ⇒ no outlier

LOF ≫1 ⇒ outlier

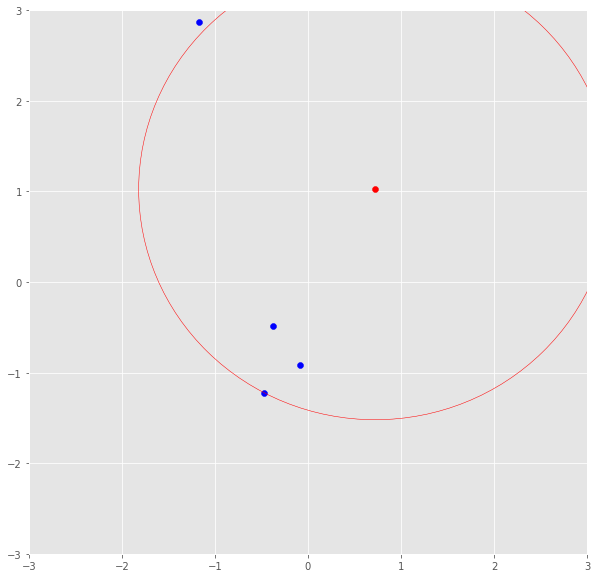
First, I introduce a parameter *k* which is the number of neighbors the LOF calculation is considering. The LOF is a calculation that looks at the neighbors of a certain point to find out its density and compare this to the density of other points later on. Using a right number *k*isn’t straight forward. While a small *k*has a more local focus, i.e. looks only at nearby points, it is more erroneous when having much noise in the data. A large *k*, however, can miss local outliers.



The density of the red point to its nearest neighbors is not different from the density to the cloud in the upper right corner. However, it is probably an outlier compared to the nearest neighbors’ density.

**k-distance**

With this *k*defined, we can introduce the *k-distance* which is the distance of a point to its *kth*neighbor. If *k* was 3, the *k-distance* would be the distance of a point to the third closest point.



The red point’s k-distance is illustrated by the red line if k=3.

**Reachability distance**

The *k-distance*is now used to calculate the reachability distance. This distance measure is simply the maximum of the distance of two points and the *k-distance* of the second point.

reach-dist(a,b) = max{k-distance(b), dist(a,b)}

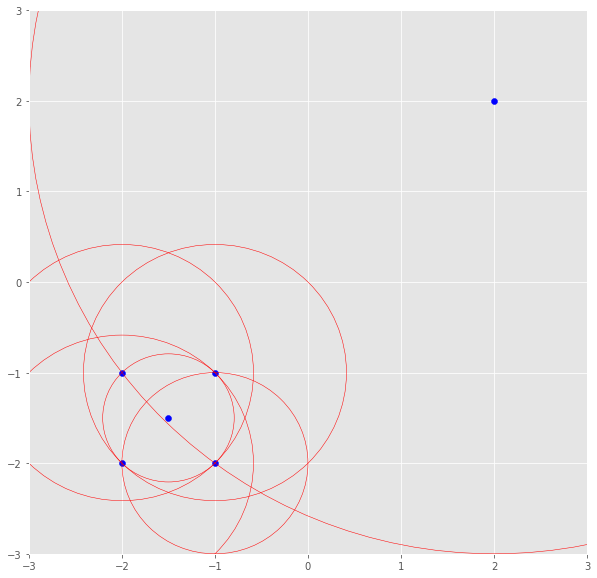
Basically, if point *a* is within the *k* neighbors of point *b*, the *reach-dist(a,b)* will be the *k-distance* of *b*. Otherwise, it will be the real distance of *a*and*b.*This is just a “smoothing factor”. For simplicity, consider this the usual distance between two points.

**Local reachability density**

The *reach-dist* is then used to calculate still another concept — the local reachability density (lrd). To get the lrd for a point *a*, we will first calculate the reachability distance of *a* to all its *k*nearest neighbors and take the average of that number. The lrd is then simply the inverse of that average. Remember that we are talking about densities and, therefore, the longer the distance to the next neighbors, the sparser the area the respective point is located in. Hence, the less dense — the inverse.

lrd(a) = 1/(sum(reach-dist(a,n))/k)

By intuition the local reachability density tells how far we have to travel from our point to reach the next point or cluster of points. The lower it is, the less dense it is, the longer we have to travel.

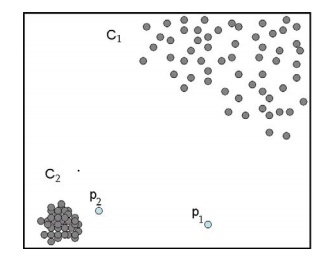


The lrd of the upper right point is the average reachability distance to its nearest neighbors which are points (-1, -1), (-1.5, -1.5) and (-1, -2). These neighbors, however, have other lrds as their nearest neighbors don’t include the upper right point.

**LOF**

The lrd of each point will then be compared to the lrd of their *k* neighbors. More specifically, *k* ratios of the lrd of each point to its neighboring points will be calculated and averaged. The LOF is basically the average ratio of the lrds of the neighbors of *a* to the lrd of *a*. If the ratio is greater than *1*, the density of point *a* is on average smaller than the density of its neighbors and, thus, from point *a*, we have to travel longer distances to get to the next point or cluster of points than from *a*’s neighbors to their next neighbors. Keep in mind, the neighbors of a point *a*may don’t consider *a*a neighbor as they have points in their reach which are way closer.

In conclusion, the LOF of a point tells the density of this point compared to the density of its neighbors. If the density of a point is much smaller than the densities of its neighbors (LOF ≫1), the point is far from dense areas and, hence, an outlier.



We have come across problems created by outliers in our model many times.If our data is one dimensional or two dimensional we can easily remove it.Did you ever thought how we can remove outlier in a 100 dimensional data?How do we find such outliers.Here comes our **Local Outlier Factor** to rescue.It helps to detect outliers in a data.Local outlier factor is a density-based method that relies on nearest neighbours search. The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors

The LOF is a calculation that looks at the neighbors of a certain point to find out its density and compare this to the density of neighbour points later on. In short we can say that the density around an outlier object is significantly different from the density around its neighbors. Let me explain how it will help.

Let us take data points **a(0,0), b(0,1), c(1,1),d(3,0).**We will find local outlier factor of each points.

# Step 1

Calculate distance between all the points.Here we will use manhattan distance.

distance(a,b) = 1  
distance(a,c) = 2  
distance(a,d) = 3  
distance(b,c) = 1  
distance(b,d) = 4  
distance(c,d) = 3

# ****Step 2****

Calculate k-distance between point z and it’s k th nearest neighbour. Here k=2 .

k-distance(a) = distance(a,c) = 2(c is the 2nd nearest neighbour)  
k-distance(b) = distance(b,a) = 1(a and c are 2nd nearest neighbour)  
k-distance(c) = distance(c,a) = 2(e is the 2nd nearest neighbour)  
k-distance(d) = distance(d,a) = 3(a and c are 2nd nearest neighbour)

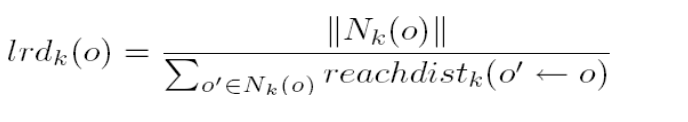
# ****Step 3****

Calculate all k-neighbours (k=2) of the each point

N(a) = {b,c}  
N(b) = {a,c}  
N(c) = {b,a}  
N(d) = {a,c}

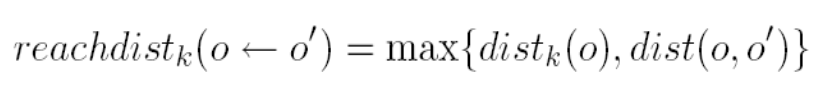
# ****Step 4****

Calculate all **local reachable densities** of each point. By intution we can say that local reachability density tells how far we have to travel from our point to reach the next point or cluster of points.The lover lrd it is less dense.

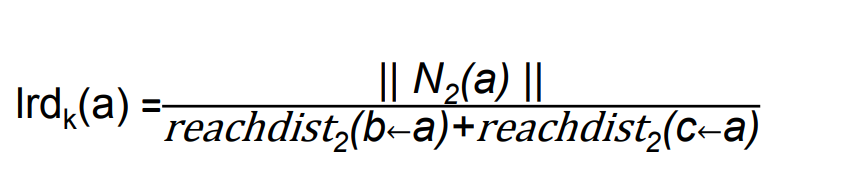


In order to find local reachable density of a point we have to find reachable distances.Here ||Nk(o)|| means number of neighbours . For example, ||Nk(a)|| = 2.

The equation for **reachable distance** is as follows:(here k=2)



Here first we will find lrd(a)



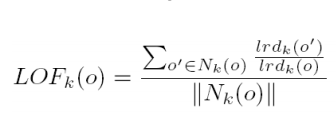
reachability distance(b,a) = max{k-distance(b),distance(a,b)}   
 = max{1,1} = 1  
reachability distance(c,a) = max{k-distance(c),distance(a,c)}   
 = max{2,2} = 2 lrd(a) = ||N(a)||  
 -------------------------------------------------------  
 reachability distance(b,a) + reachability distance(c,a)  
   
 = 2/(1+2) = 0.667

Similarly we can find local reachable density of b,c and d

lrd(b) = 2/(2+2) = 0.5  
lrd(c) = 2/(1+2) = 0.667  
lrd(d) = 2/(3+3) = 0.33

# ****Step 5****

Calculate **local outlier factor**



**LOF(a)** = (lrd(b) + lrd(c))  
 -----------------  
 ||N(a)||\*(lrd(a))  
   
 = (0.5 + 0.667) /(2\*0.667) = 0.8748

Similarly we calculate local outlier factor of b, c and d.

LOF(b) = (lrd(a) + lrd(c))  
 -----------------  
 ||N(b)||\*(lrd(b)) = (0.667+0.667) /(2\*0.5) = 1.33LOF(c) = (lrd(b) + lrd(a))  
 -----------------  
 ||N(c)||\*(lrd(c))  
   
 = (0.5+0.667) / (2\*0.667) = 0.8748LOF(d) = (lrd(a) + lrd(c))  
 -----------------  
 ||N(d)||\*(lrd(d)) = (0.667+0.667) /(2\*0.33) = 2

# Step 6

Sort all LOF

LOF(d) = 2  
LOF(b) = 1.33  
LOF(a) = 0.8748  
LOF(c) = 0.8748

The next question that comes to our mind is which point is outlier? Actually it depends on data.We can say Point d is the first outlier.In general we can say that if LOF value is small (LOF <1),it can be an inlier and if it is large,it can be outlier(LOF >>1).

# ****Implementation using Python****

Here we will implement the same using python. We will use **LocalOutlierFactor** from **sklearn** for that.

import numpy as np  
from sklearn.neighbors import LocalOutlierFactor  
x = np.array([[0,0],[0,1],[1,1],[3,0]])  
clf = LocalOutlierFactor(n\_neighbors=2,metric='manhattan')  
y\_pred = clf.fit\_predict(x)  
scores = clf.negative\_outlier\_factor\_  
print(-scores)

Output:

[0.875 1.33333333 0.875 2. ]

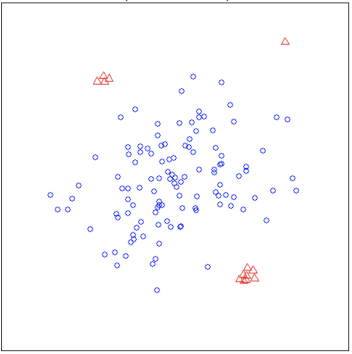
Here we print(-scores) because by default negative\_outlier\_factor gives negative value of the same.To change to positive we used print(-score)

# Isolation forest method

Isolation forest detects anomalies by randomly partitioning the domain space. Yeah, you’re heard me right- It works similar to [Decision trees](https://scikit-learn.org/stable/modules/tree.html) algorithm, where we start with a root node and keep on partitioning the space. In Isolation forest we partition randomly, unlike Decision trees where the partition is based on [Information gain](https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1).

Partitions are created by randomly selecting a feature and then randomly creating a split value between the maximum and the minimum value of the feature. We keep on creating the partitions until we isolate all the points(in most cases we also set a limit on number of partitions/height of the tree).

Now let us visualise how a normal point will differ from an anomalous points in our feature space.



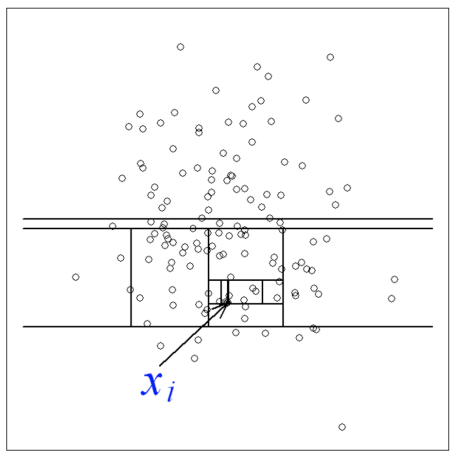
**Visualising anomalies**

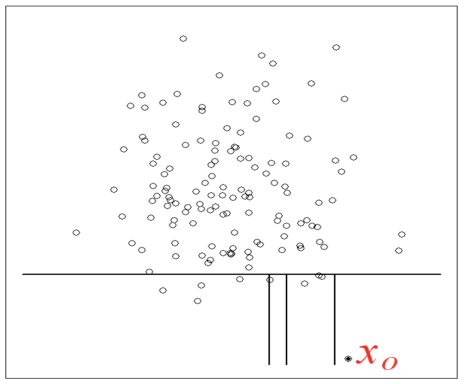
In the above image, the red points denote the anomalous points whereas the blue ones denotes the normal points.

Now clearly, a normal point will be more clustered while an anomalous point will be far away from other points. Thus while randomly partitioning the domain space, the anomaly will be detected in smaller number of partitions than a normal point. Smaller number of partitions means lesser is the distance from the root node(this means lesser number of edges need to be travelled from root node to the terminal node).

The above discussed concept will be well evident with the images given below.

**Isolating a normal point**





**Isolating an anomalous point**

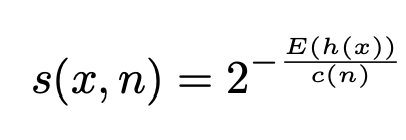
*the anomaly will be detected in smaller number of partitions than a normal point*

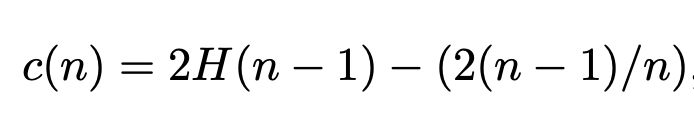
So clearly the path length indicates whether a point is a normal or an anomalous point. (**Path length-**of a point x is measured by the number of edges x traverses an Isolation tree from the root node until the traversal is terminated at an external node)

Isolation forest is an ensemble method. So we create multiple Isolation trees(generally 100 trees will suffice) and we take the average of all the path lengths.This average path length will then decide whether a point is anomalous or not.

**Anomaly score-**

Anomaly score is given by the following formula-





where

**n-**Number of data points

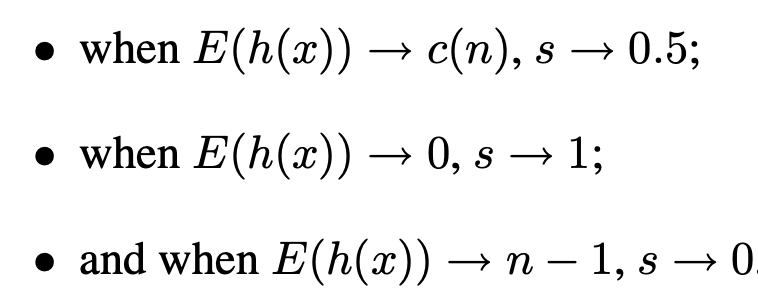
**c(n)**- It is the average path length of unsuccessful search in a Binary search tree. (Feel free to go through Pages 660, 661 in this [book](http://programming.etherealspheres.com/backup/ebooks%20-%20Other%20Programming%20Languages/Data%20Structures%20And%20Algorithms%20With%20Object-Oriented%20Design%20Patterns%20In%20C%20Sharp.pdf) for the derivation)

Now observe. We grow an isolation tree by randomly choosing a feature and randomly partitioning. This is very similar to the [Binary Search tree](https://www.geeksforgeeks.org/binary-search-tree-data-structure/). Thus we can approximate the average path length of for a node termination with the unsuccessful search in a Binary Search tree. Thus we use c(n) as the reference.

***If you’re having a hard time figuring out about Binary search trees, c(n) is just a reference metric. It normalises the score between 0 to 1.***

Note that it is always better to represent score between 0 to 1 because the score can now be interpreted as a probability. For example, say for a data point if we get the anomaly score as 0.8, then we can interpret such that the point has a probability of 80% to be an anomalous point.

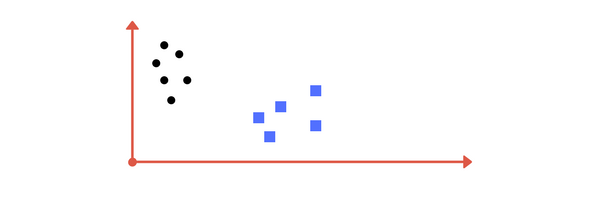
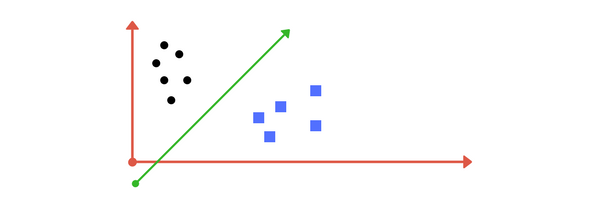
**E(h(x))**- Average of path lengths from the Isolation forest



* As score is closer to 1, then it is an anomalous point
* As the score is closer to 0, it a normal observation
* A score near 0.5, indicates it doesn’t have much distinction from normal observations

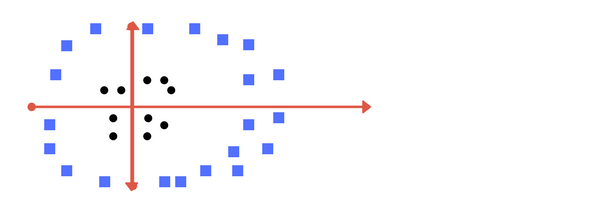
**SVM**

0. Introduction

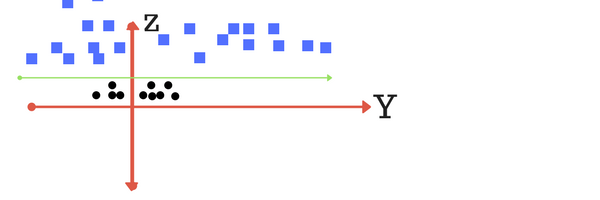
* *A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.*
* **Confusing? Don’t worry, we shall learn in laymen terms.**
* Suppose you are given plot of two label classes on graph as shown in image (A). Can you decide a separating line for the classes?
* 
* Image A: Draw a line that separates black circles and blue squares.
* You might have come up with something similar to following image (image B). It fairly separates the two classes. Any point that is left of line falls into black circle class and on right falls into blue square class. **Separation of classes. That’s what SVM does.** It finds out a line/ hyper-plane (in multidimensional space that separate outs classes). Shortly, we shall discuss why I wrote multidimensional space.
* 
* s Image B: Sample cut to divide into two classes.

# 1. Making it a Bit complex…

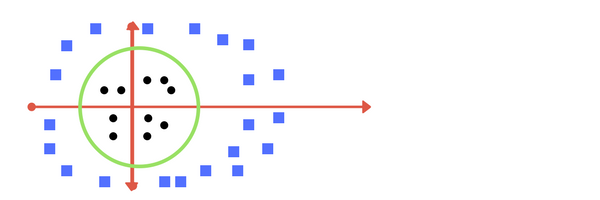
* So far so good. Now consider what if we had data as shown in image below? Clearly, there is no line that can separate the two classes in this x-y plane. So what do we do? We apply transformation and add one more dimension as we call it z-axis. Lets assume value of points on z plane, w = x² + y². In this case we can manipulate it as distance of point from z-origin. Now if we plot in z-axis, a clear separation is visible and a line can be drawn .



* Can you draw a separating line in this plane?



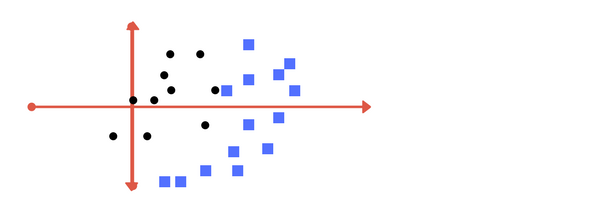
* plot of zy axis. A separation can be made here.
* When we transform back this line to original plane, it maps to circular boundary as shown in image E. These transformations are called **kernels.**



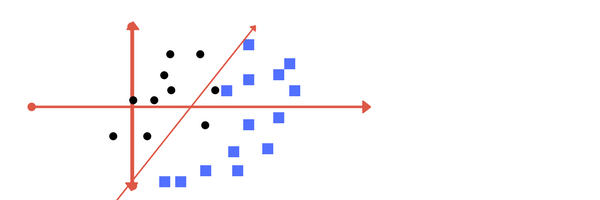
* Transforming back to x-y plane, a line transforms to circle.
* *Thankfully, you don’t have to guess/ derive the transformation every time for your data set. The sklearn library's SVM implementation provides it inbuilt.*

# 2. Making it a little more complex…

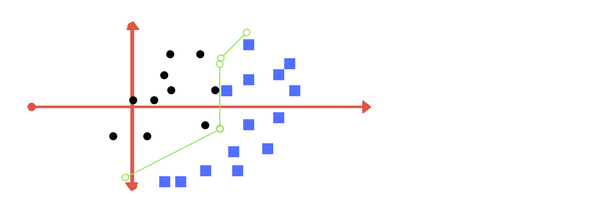
* What if data plot overlaps? Or, what in case some of the black points are inside the blue ones? Which line among 1 or 2?should we draw?



* What in this case?



* **Image 1**



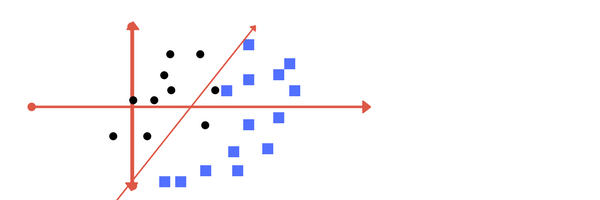
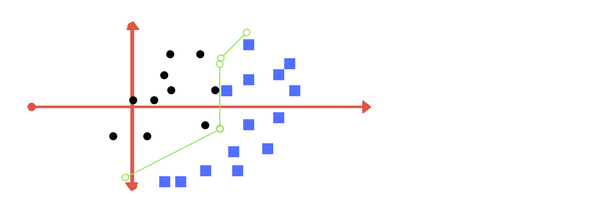
* **Image 2**
* Which one do you think? Well, both the answers are correct. The first one tolerates some outlier points. The second one is trying to achieve 0 tolerance with perfect partition.
* **But, there is trade off**. In real world application, finding perfect class for millions of training data set takes lot of time. As you will see in coding. This is called **regularization parameter**. In next section, we define two terms **regularization parameter**and **gamma.**These are tuning parameters in SVM classifier. Varying those we can achive considerable non linear classification line with more accuracy in reasonable amount of time. In coding exercise (part 2 of this chapter) we shall see how we can increase the accuracy of SVM by tuning these parameters.
* One more parameter is **kernel.**It defines whether we want a linear of linear separation. This is also discussed in next section.
* 
* When somebody asks me for advice.

# 3. Tuning parameters: Kernel, Regularization, Gamma and Margin.

# ****Kernel****

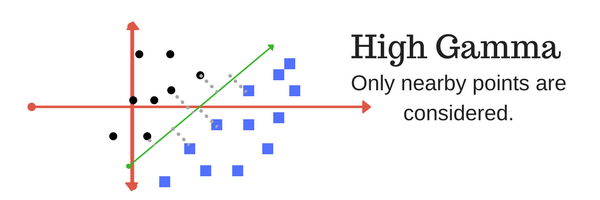
* The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.
* For **linear kernel** the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:
* f(x) = B(0) + sum(ai \* (x,xi))
* This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.
* The **polynomial kernel** can be written as K(x,xi) = 1 + sum(x \* xi)^d and **exponential** as K(x,xi) = exp(-gamma \* sum((x — xi²)). [Source for this excerpt : <http://machinelearningmastery.com/>].
* *Polynomial and exponential kernels calculates separation line in higher dimension. This is called****kernel trick***

# Regularization

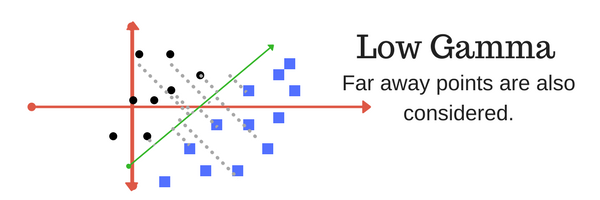
* The Regularization parameter (often termed as C parameter in python’s sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.
* For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of 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

* 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.

