Random Forest
Support Vector Machine

Lecture # 12

Bagging and Random Forest

Bagging and Random Forest are ensamble learning methods (the idea of combining the outputs of multiple models through some kind of voting or averaging).Bagging averages the predictive results of a series of bootstrap samples from a training set of data. Often applied to decision trees, bagging is applicable to regression and many nonlinear model fitting or classification techniques. For a sample of N points in a training set, bagging generates K equal sized bootstrap samples from which one could estimate f_i (x) with the final estimator given as the average of f_i (x) values.

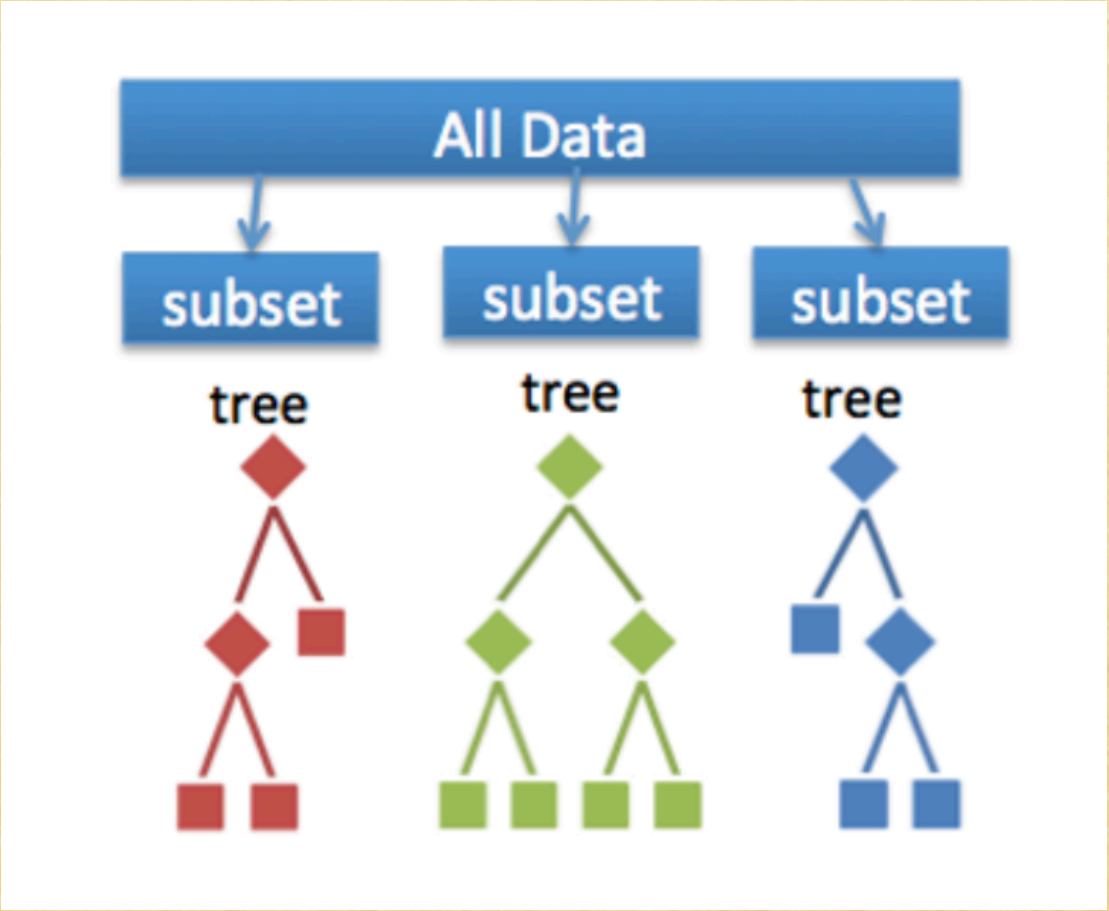
Random forest generates a series of decision trees from these bootstrap samples. The features on which to generate the trees is selected at random from the full set of features in the data. The final classification from random forest is based on the averaging of the classifications of each of the individual decision trees. Random forest addresses limitations of decision trees: overfitting of the data.

To Build a Random Forest

To generate a random forest we define *n*, the number of trees that we will generate, and *m*, the number of attributes that we will consider splitting on at each level of the tree. For each decision tree a subsample of the data is selected from the full dataset (bootstrap sampling). At each node of the tree, a set of m variables are randomly selected and the split criteria is evaluated for each of these attributes (different set of *m* attributes for each node). The classification is derived from the mean or mode of the results from all the trees. By keeping *m* small compared to the number of features controls the complexity of the models and reduces overfitting.

In simple words, Random forest builds multiple decision trees (called the forest) and glues them together to get a more accurate and stable prediction. The forest it builds is a collection of Decision Trees, trained with the bagging method.

Random Forest



The Difference between Decision Tree and Random Forest

Example: Suppose you are planing to buy a house. The range of important parameters you consider for buying the house are:

Price of the house

Locality

Number of bedrooms,

Parking space

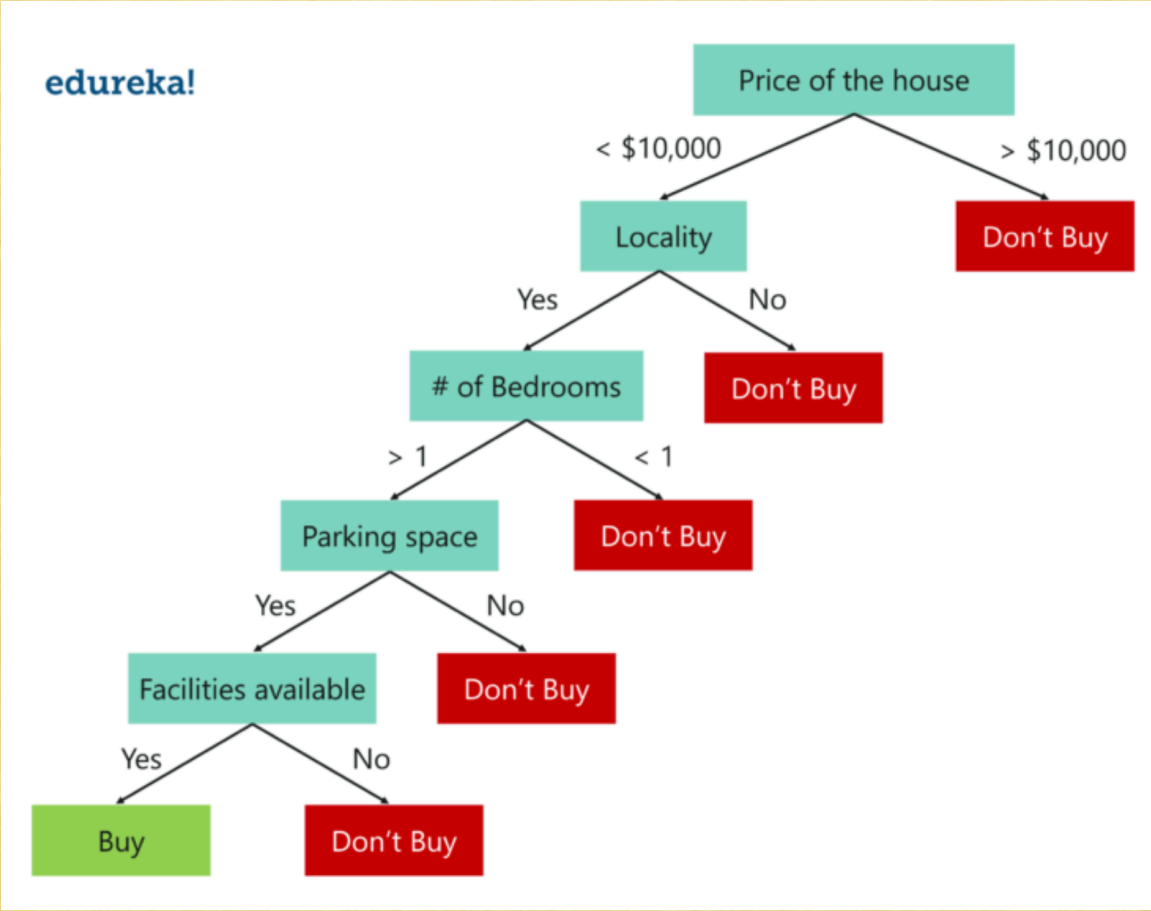
Available facilities

These parameters are called predictor variables, which are used to find the response variables. Using these parameters, we could build a decision tree. Decision trees are built on the entire parameters space.

Random forest is an ensemble of decision trees. It randomly selects a set of parameters and creates a decision tree for each set of chosen parameters

Example of a Decision Tree when using all the available Data

(From edureka webpage)

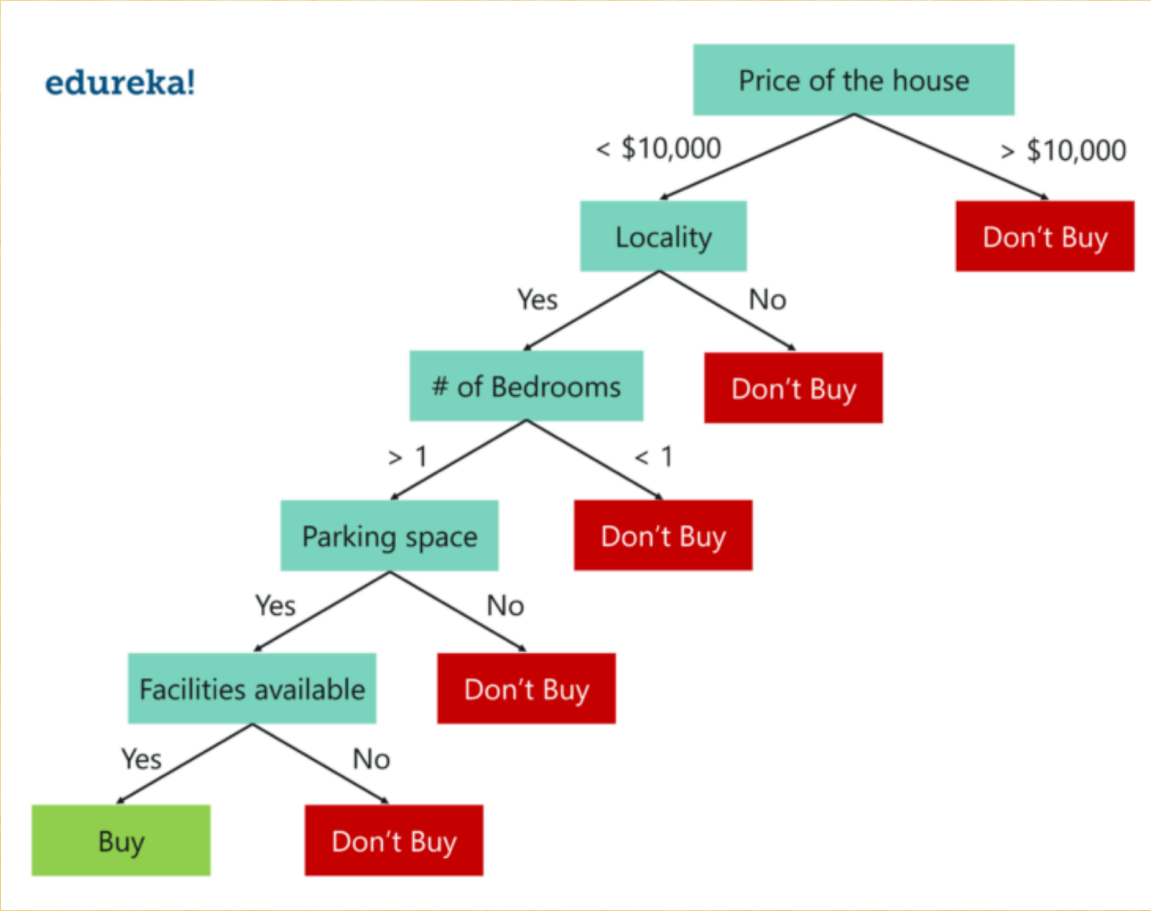


Random forest is an ensemble of decision trees. It randomly selects a set of parameters and creates a decision tree of each set of chosen parameters.

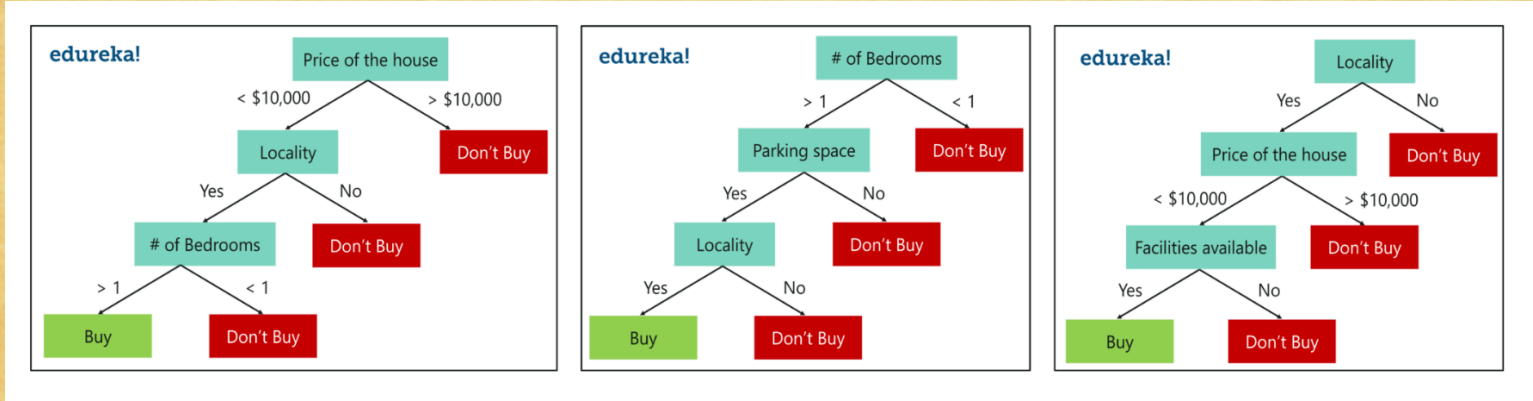
Next figure shows a set of three decision trees with each tree taking only three parameters from the entire dataset. Each tree predicts the outcome based on the respective predictor variables used in that tree and takes the average of the results from all the decision trees in the random forest.

After creating multiple Decision trees using this method, each tree selects or votes the class (in this case the decision trees will choose whether or not a house is bought), and the class receiving the most votes by a simple majority is termed as the predicted class.

Decision tree using all the parameters



Three Decision Trees based on sub-sets of the parameters



Advantages of Random Forest compared to Decision Trees

What random forest has that decision trees do not?

Even though Decision trees are convenient and easily implemented, they lack accuracy. Decision trees work very effectively with the training data that was used to build them, but they're not flexible when it comes to classifying the new sample. Which means that the accuracy during testing phase is very low. This happens due to a process called Over-fitting.

Over-fitting occurs when a model studies the training data to such an extent that it negatively influences the performance of the model on new data.

This means that the disturbance in the training data is recorded and learned as concepts by the model. But the problem here is that these concepts do not apply to the testing data and negatively impact the model's ability to classify the new data, hence reducing the accuracy on the testing data.

This is where Random Forest comes in. It is based on the idea of bagging, which is used to reduce the variation in the predictions by combining the result of multiple Decision trees on different samples of the data set.

Support Vector Machine

Definition and Terminologies

A support vector machine (SVM) is a classification method. It works on the principle of fitting a boundary to a region of points that are all alike (belonging to the same class). Once a boundary is fitted on a training sample, for any new points (test sample) that need to be classified, one must check if they lie inside the boundary or not. What is needed here is a set of points that can help to fix the boundary. These data points are called support vectors because they support the boundary. Why are they called vectors? Because each data point or observation is a vector: that is, it is a row of data that contains values for a number of different attributes.

In 2-D plots, the line separating the data is just a line. In 3-D this is a plane separating the data. This could be extended to higher dimensions. The plane separating data in high dimensions is called a hyperplane. The hyperplane is our decision boundary. Everything on one side belongs to one class and everything on the other side belongs to a different class.

When the two classes are completely separated by drawing a linear line between them, it is called linearly separable.

The decision boundary should be optimized so that the data from different classes are as far away from it as possible. The farther the data are from the boundary more confident we are regarding the classification (Figure 4). In some cases more than one decision boundary is found. The optimum is the one that maximizes distances from all the points from the two classes. In this case, the points have larger margins and therefore, classifications would be more reliable. The aim here is to find the points closest to the separating hyperplane and make sure this is as far apart from the separating line as possible. This is known as margin. We want to have the greatest possible margin. This is because if we make a mistake or trained our classifier on limited data, we want it to be as robust as possible. The points closest to the separating hyperplane are known as support vectors. Therefore, the problem reduces to maximizing the distance from the separating line to support vectors.

Summary

A support vector machine constructs a hyperplane or a set of hyperplanes in high dimensional space which can be used for classification, regression or outlier detection. The best separation is achieved by the hyperspace that has the largest distance to the nearest training data point of any class (called margin). The larger the margin, the smaller is the error in the classifier.

Linearly separable

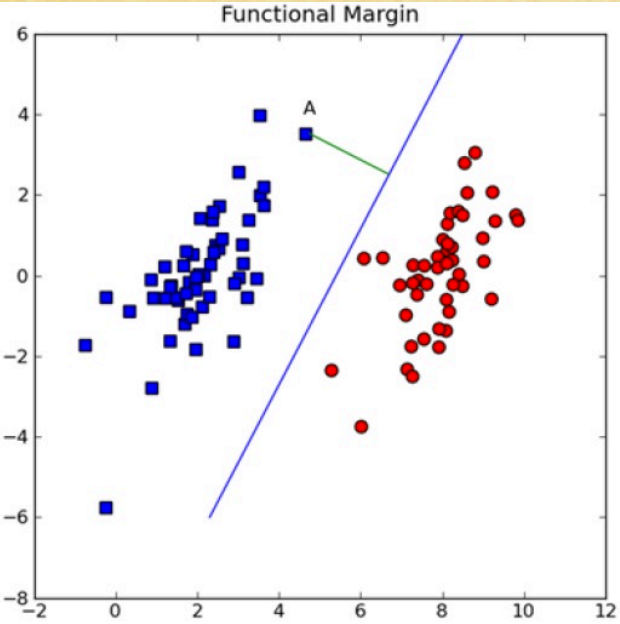


Figure 6.3 The distance from point A to the separating plane is measured by a line normal to the separating plane.

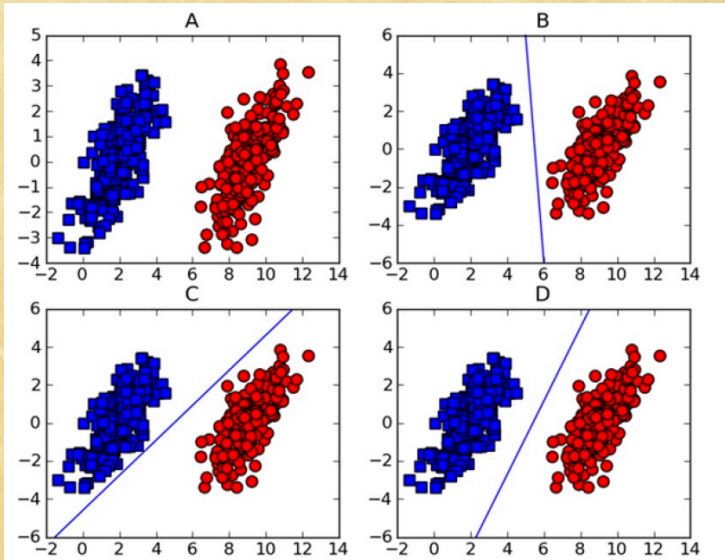


Figure 6.2 Linearly separable data is shown in frame A. Frames B, C, and D show possible valid lines separating the two classes of data.

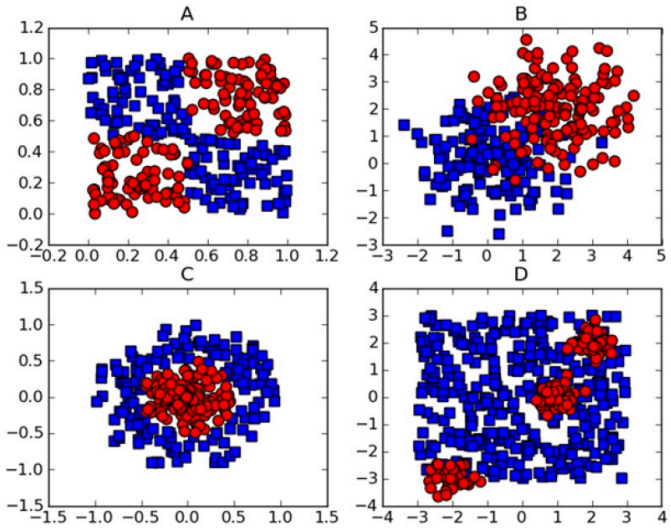
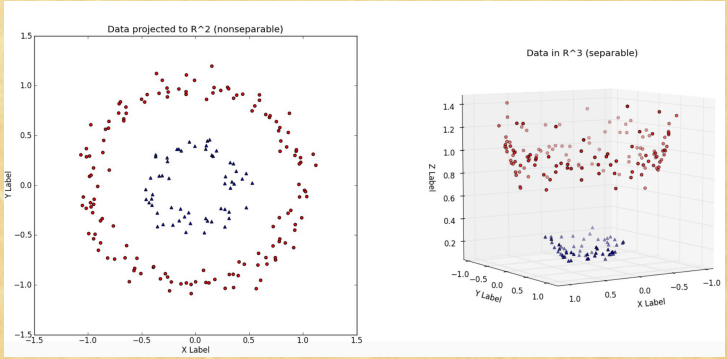
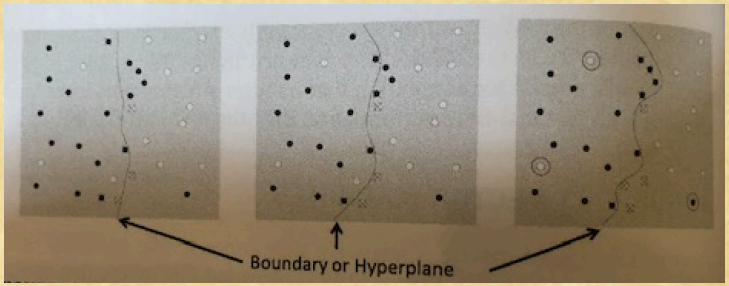


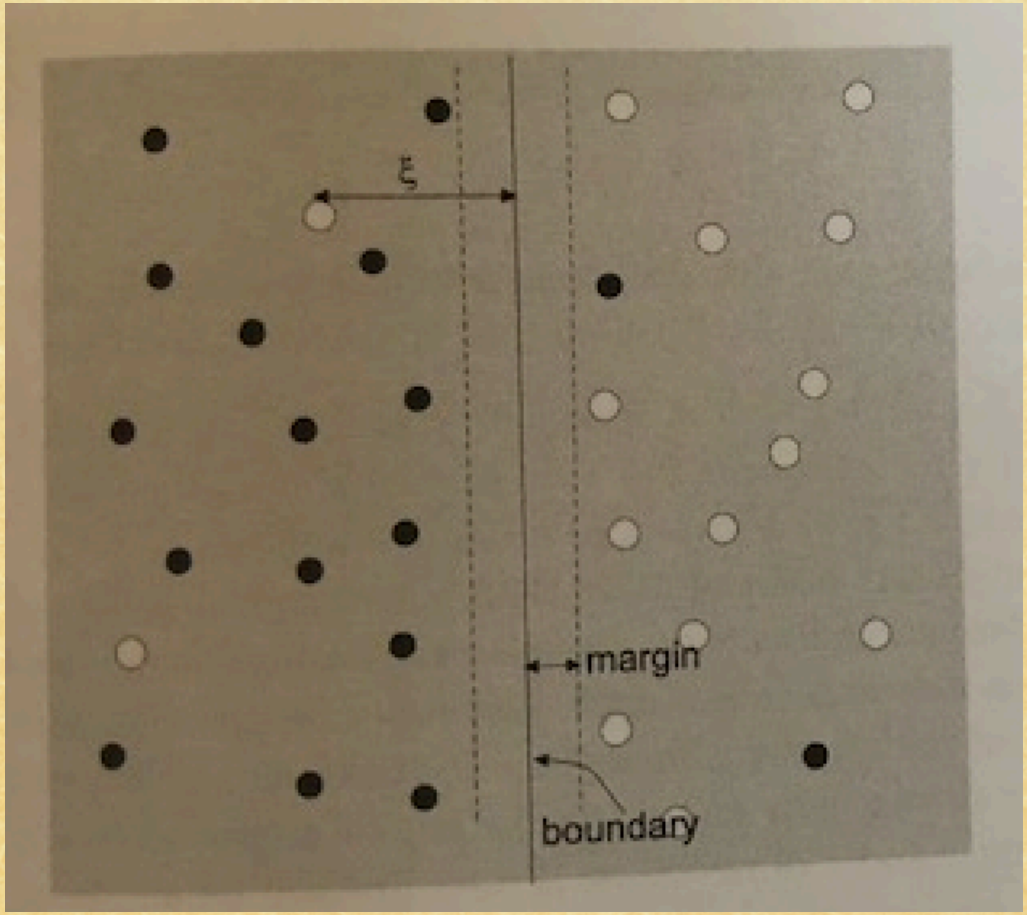
Figure 6.1 Four examples of datasets that aren't linearly separable



It is not always possible to ensure that the data are cleanly separable and rare to find that the data are linearly separable. When this happens there may be many points within the margin. In this case the best hyperplane is the one that has the minimum number of such points within the margin. To insure this, a penalty is charged for every contaminant inside the margin and the hyperplane that has the minimum penalty is chosen. In the next Figure ξ represents the penalty that is applied for each error and the sum of all such errors is minimized to get the best separation.

In this Figure a number of hyperplanes can be found to separate the same data set. The boundary that separates the classes with minimum misclassification is the best. In this data set, the algorithm that applies to the third plot has zero misclassifications and therfore is the best one. Additionally, a boundary line that ensures that the average geometric distance between the two regions (or classes) is maximized, is even better. This n-dimensional distance is called a margin




ξ
margin
boundary

We are given a training dataset of n points of the form

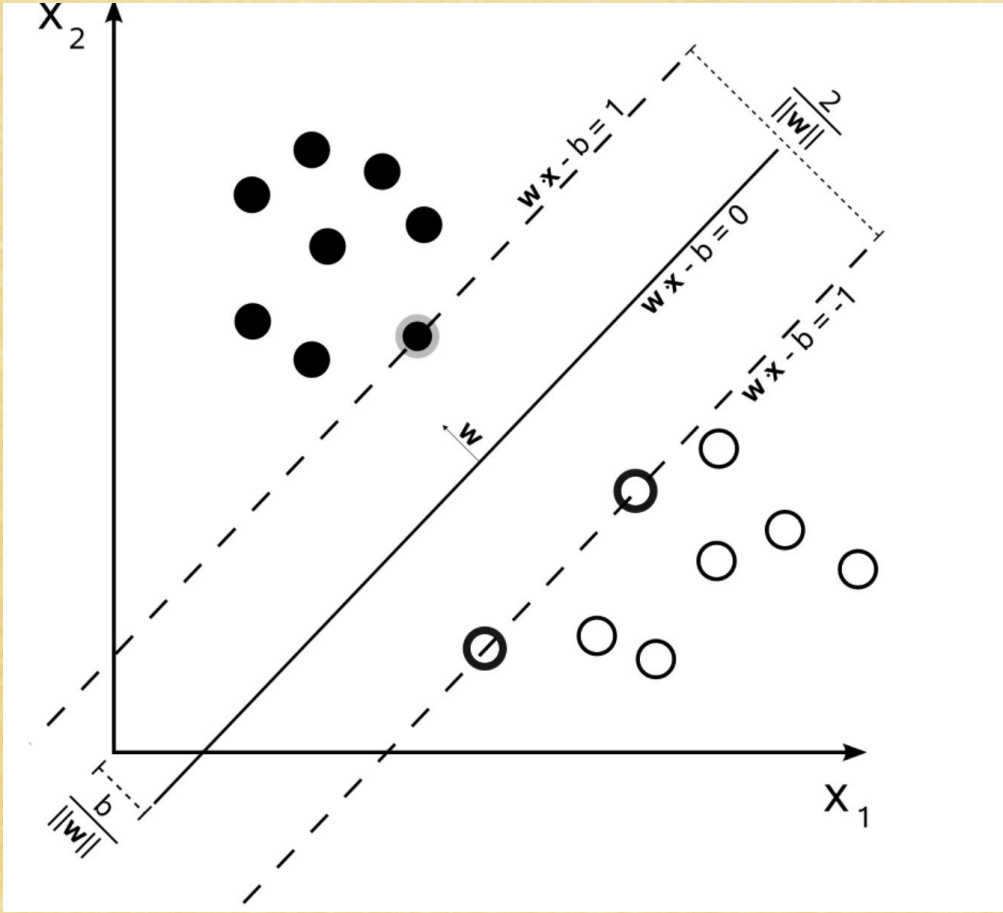
$(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)$ where y_i are either 1 or -1, each indicating the class

to which the vector x_i belongs. Each x_i is a p-dimensional real vector. We want to find the maximum margin hyperplane that divides the group of points x_i for which $y_i=1$ to which $y_i= -1$ which is defined so that the distance between the hyperplane and the nearest point x_i from either group is maximized.

A hyper plane can be written in the form $\vec{w}.\vec{x} - b = 0$ where $\vec{w}$

is the normal vector to the hyperplane. The offset of the hyperplane from the origin along the normal vector $\vec{w}$ is $\frac{b}{\|w\|}$

Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are the support vectors.



How SVM Works

If the training data is linearly separable, we can select two parallel hyperplanes that separates the two classes of data, so that the distance between them is as large as possible. The region between these two hyperplanes is called the margin and the maximum margin hyperplane is the hyperplane that lies halfway between them. These hyperplanes can be described by the equations

$$\vec{w} \cdot \vec{x} - b = 1 \ \ (anything\ on\ or\ above\ this\ boundary\ is\ of\ one\ class - class\ 1$$

and

$$\vec{w} \cdot \vec{x} - b = -1\ (anything\ on\ or\ below\ this\ boundary\ is\ other\ class - class - 1$$

The distance between these two hyperplanes is $\frac{2}{\|w\|}$.

To maximize the distance between the planes, we need to minimize $\|\vec{w}\|$.

The distance is computed using the equation for distance from a point to a plane. We also need to avoid data points falling in the margins. Therefore, we add the following constraint.

For each *i*

$$\vec{w} \cdot \vec{x} - b \geqq 1 \ \ if \ y_i = 1$$

or

$$\vec{w} \cdot \vec{x} - b \leqq 1 \ if \ y_i = -1$$

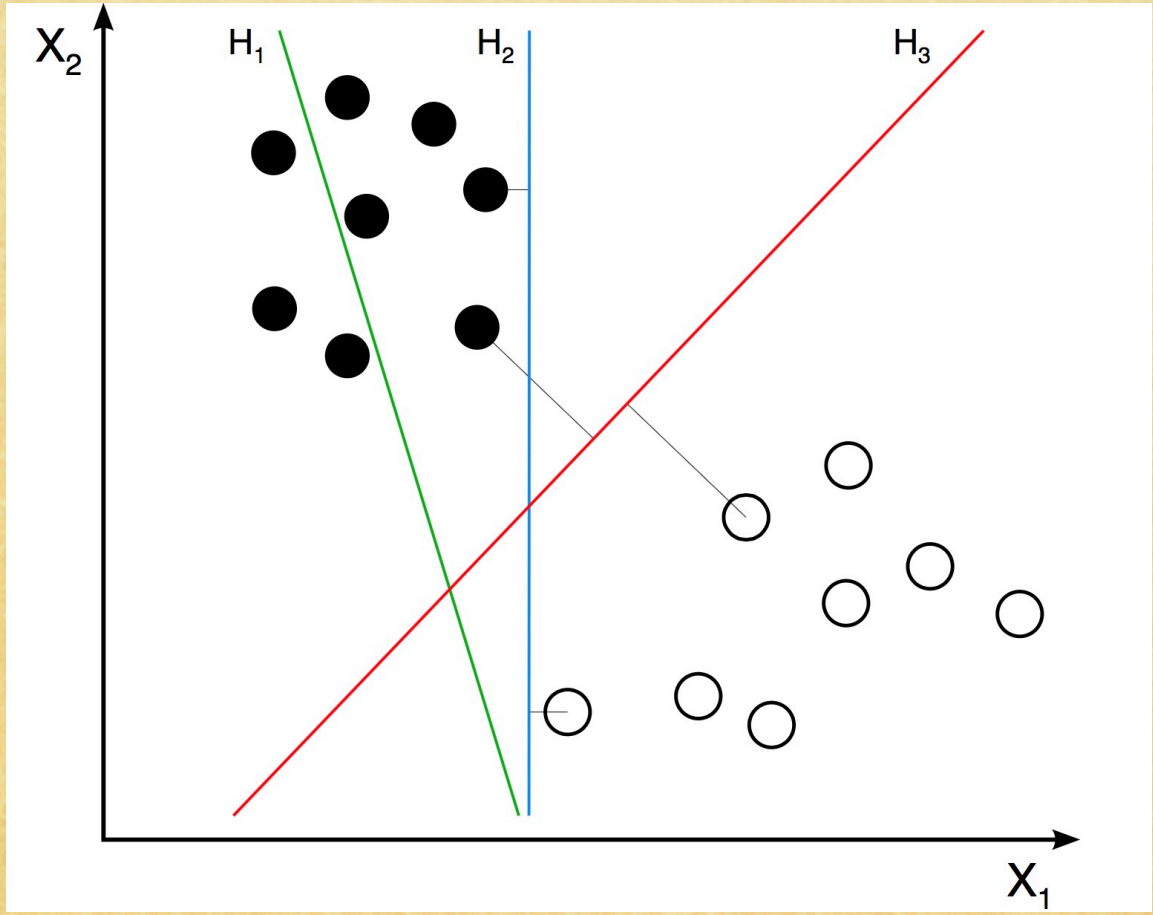
These constraints state that each data point must lie on the correct side of the margin. Therefore, this can be re-written as

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geqq 1 \ \ for \ all \ 1 \leqq i \leqq n$$

We can now define the optimization problem as to minimize $\| \vec{w} \|$ subject to

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geqq 1, for \ i = 1, \ldots, n.$$

The w and b that optimizes this, determines our classifier. It is clear that the maximum margin hyperplane is completely determined by those x_i values that lie nearest to it. These x_i 's are called support vectors.


X2
H1
H2
H3
X1

In most cases the data are not linearly separable. In such circumstances we form the function

$$\max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b))$$

Note that y_i is the ith target (-1 or 1) and $(\vec{w} \cdot \vec{x}_i - b)$ is the current output. This function is zero if the first constraint above is satisfied- in other words, if x_i lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin. We then minimize

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b))\right] + \lambda \|\vec{w}\|^2$$

Where the parameter *i* determines the trade-off between increasing the margin size and ensuring that the x- lie on the correct side of the margin. For small values of *i*, the second term in the right hand side of the equation becomes negligible and therefore, it will behave similar to the previous case if the input data are linearly classifiable.

Computing SVM

To compute SVM, one needs to minimize the expression

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\vec{w}.\vec{x}_i - b))\right] + \lambda \|\vec{w}\|^2$$

There are different ways to minimize this function

Minimization Methods

Method 1:

This reduces the function to a quadratic programming problem. This is a method that optimizes (minimize or maximize) a quadratic function of several variables subject to linear constraints on these variables.

For each $i \in \{1, \ldots, n\}$ we introduce variable $\zeta_i = \max(0, 1 - y_i(w.x_i - b))$ with ζ_i being the smallest nonnegative number satisfying $y_i(w.x_i - b) \geq 1 - \zeta_i$.
Therefore, we can write the optimization problem as

$$minimize \frac{1}{n} \sum_{i=1}^{n} \zeta_i + \lambda \|\vec{w}\|^2$$

Subject to $y_i(w.x_i - b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$ for all i values. This is called the primal problem.

Method 2:

The other method to minimize the function

$$f(w,b) = \left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\vec{w}.\vec{x}_i - b))\right] + \lambda \|\vec{w}\|^2$$

Is sub-gradient descent algorithm. Here f (w,b) is a convex function of w and b . Therefore traditional gradient descent method can be adopted where instead of taking a step in the direction of the functions gradient, an step is taken in the direction of a vector selected from the function's sub-gradient.

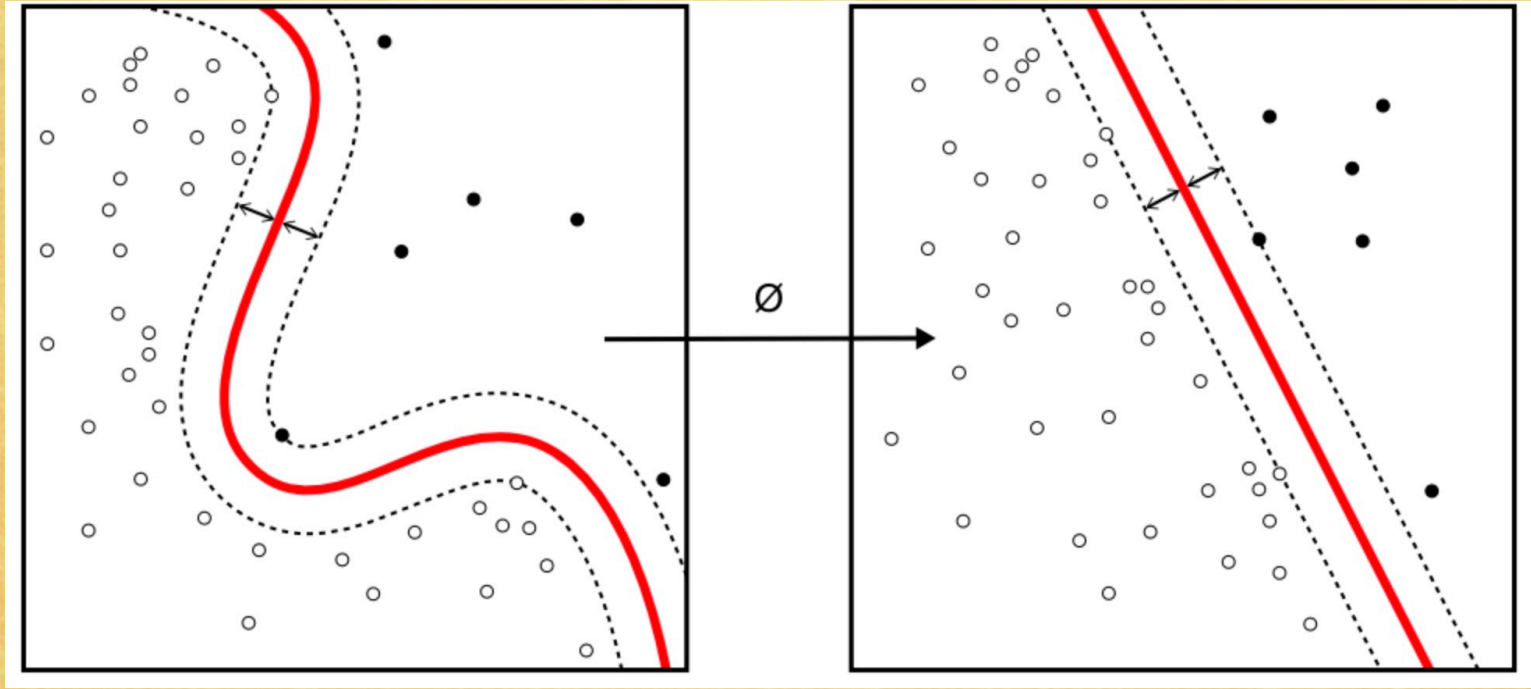
Kernel Functions

Kernel functions provide the option of transforming non-linear space into linear ones. There are a range of non-linear kernels from simple polynomial basis functions to sigmoid functions. We need to choose the appropriate kernel function.

With a large number of attributes in a dataset, it is hard to know which kernel would work best. The most commonly used ones are polynomials. It is often a good idea to start with a quadratic polynomial and move to using some exotic kernel functions until the required accuracy is reached.

While the original problem can be stated in a finite dimensional space, sometimes the sets that discriminate are not linearly separable in that space. For this reason the original finite dimensional space is mapped into a much higher dimensional space, making the separation easier in that space. The mapping used by SVM mechanism is designed to ensure that dot products can be computed in terms of variables in the original space by defining them in terms of a kernel function k(x,y)- (Figure 6). The hyperplanes in higher dimensional space are defined as the set of points whose dot products with a vector in that space is constant. The vectors defining the hyperplanes are chosen to be linear combinations with coefficients α_i of images of feature vectors x_i that occur in the data base. With this choice of a hyperplane, the points x in the feature space that are mapped into the hyperplane are defined by the relation

$$\sum_i \alpha_i k(x_i, x) = constant$$


Ø

Kernel Trick

What do we do if the data are not linearly separable? We use Kernels. The data that are not separable in n dimensional space may be linearly separable in higher dimensional space.

Lets say the decision boundary. i.e the hyperplane separating the classes have the weights (Co-coefficients) given by vector w. This is what we need to find. We try to maximize the distance from this w vector to the nearest points (support vectors) so this now becomes our constraint. Rewriting the equation

$$L = \frac{1}{2}\|w\|^2 - \sum_i \alpha_i [y_i . (\bar{w} . \bar{x}_i + b) - 1]$$

We minimize this equation

$$\frac{\partial L}{\partial \bar{w}} = \bar{w} - \sum_i \alpha_i y_i x_i = 0$$

where

$$\bar{w} = \sum_i \alpha_i y_i x_i$$

and

$$\frac{\partial L}{\partial b} = - \sum_i \alpha_i \, y_i = 0$$

$$L = \frac{1}{2} (\sum_i \alpha_i y_i x_i) (\sum_j \alpha_j y_j x_j) - (\sum_{\square} \alpha_i y_i x_i \sum_j \alpha_j y_j x\_j) - \sum_i \alpha_i y_i b + \sum_i \alpha_i$$

Therefore

Therefore,

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_j$$

i counts the number of data points in our training data. y's denote the outputs of the data points, which for our convenience, we express as +1 or -1. x is the feature vector in each training example. α is the constraints or Lagrangian multipliers. Lagrangian multipliers are used to include the constraints for solving a minimization or maximization problems, thus enabling us to not worry about them while reaching the solution.

While minimizing for W(alpha) [Weight vector as a function of alpha] we see the term x*x(transpose). That is to say that we do not exactly need the exact data points, but only their inner products to compute our decision boundary.

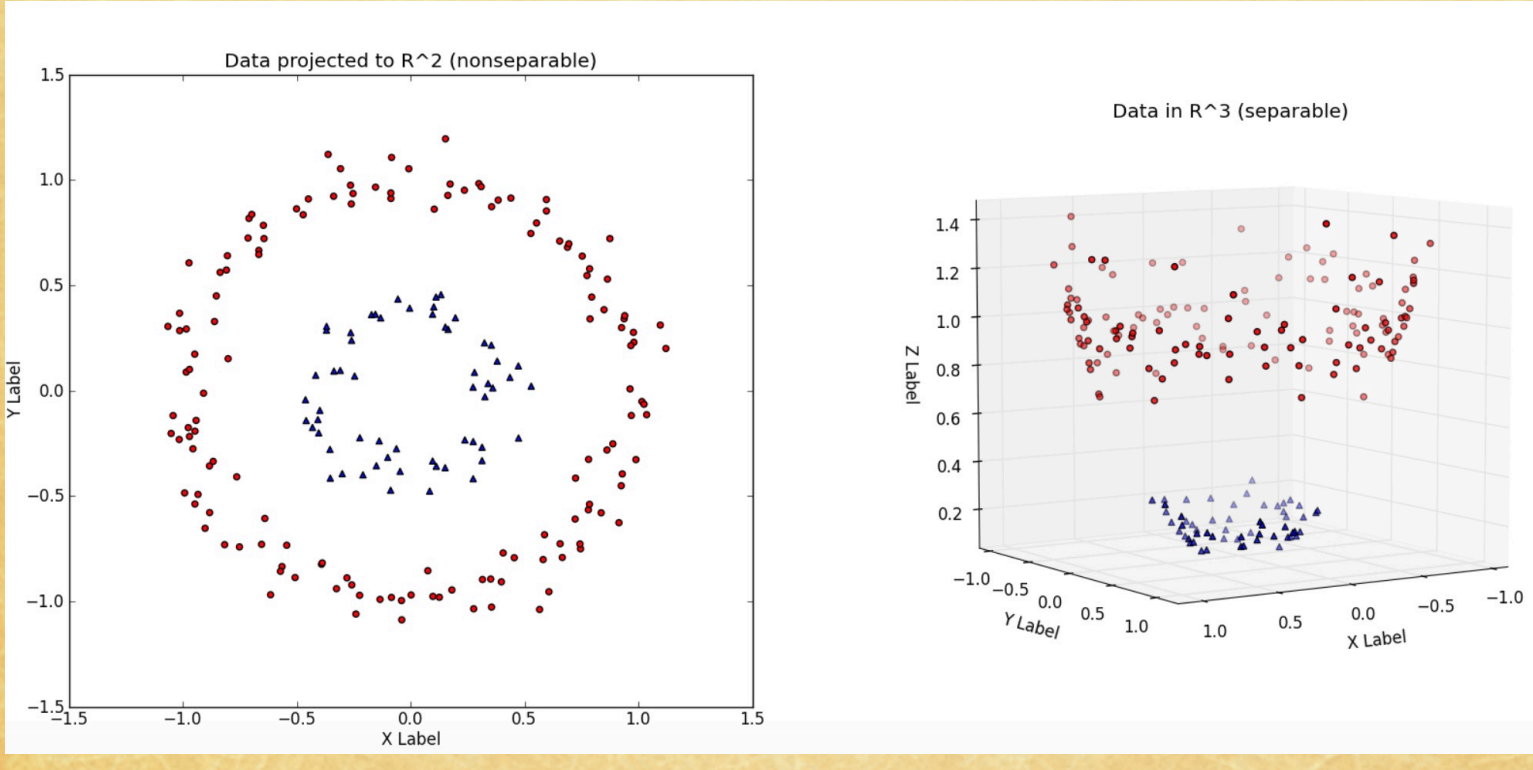
What it implies is that if we want transform our existing data into a higher dimensional data, which in many cases help us classify better (see the image below for an example) we need not compute the exact transformation of our data, we just need the inner product of our data in that higher dimensional space. This works for datasets which aren't linearly separable!

It's a lot easier to get the inner product in a higher dimensional space than the actual points in a higher dimensional space. Polynomial kernels by simply using exponents of 'd' to map our data into a 'd' dimensional space can be effective for our solution.

Summary

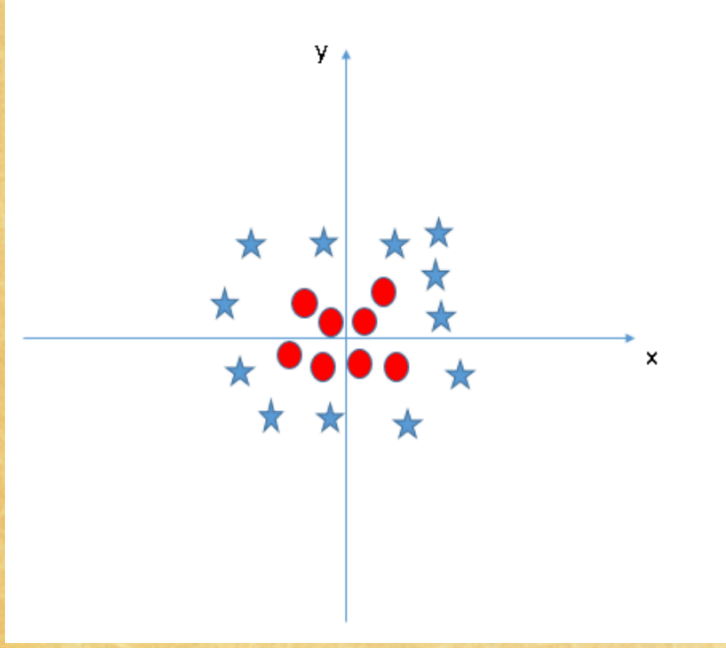
The Kernel Trick is a technique in SVM. These are functions which take low dimensional input space and transfer them into a higher dimensional space. It convers problems that are not separate (and hence complex) to separate problem (simpler to solve). It carries on data transformation and finds out the process that would separate the data, based on the labels and outputs defined.

By transforming data points from lower to higher dimension, we change the data to linearly separable

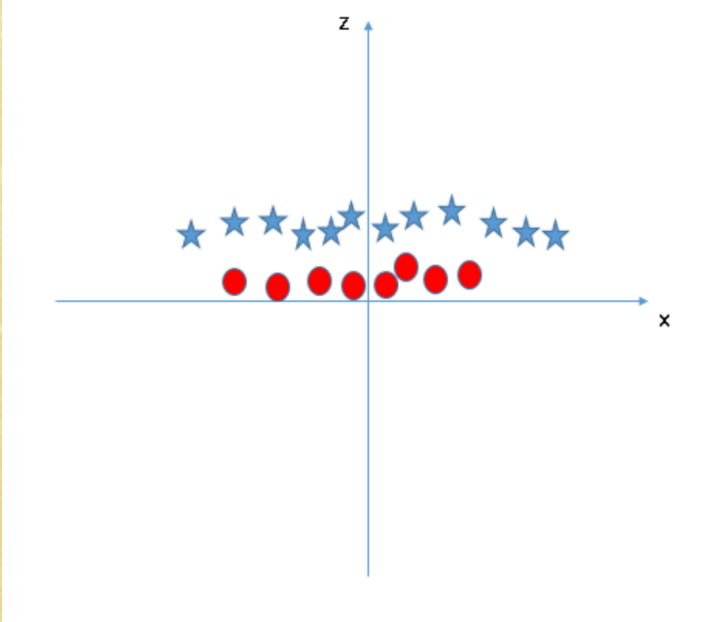


Kernel Trick

It is hard to fit a hyperplane to separate these data



Define $z=(x^2+y^2)^{1/2}$ that separates the data on x-z space



Sources used for this lecture

1. Machine Learning In Action

By Peter Harrington

2. Data Science: Concepts and Practice

By Vijay Kotu Bala Deshpande