SVM –Support Vector Machines

Derive the equation for the Maximal Margin Hyperlane (MMH) as a decision boundary in case of an SVM classifier.

Ans: *Support vector machines (SVMs)* are naturally defined for binary classification of numeric data. Categorical feature variables can also be addressed by transforming categorical attributes to binary data with the binarization approach. It is assumed that the class labels are drawn from $\{-1, 1\}$. As with all linear models, SVMs use separating hyperplanes as the decision boundary between the two classes. In the case of SVMs, the optimization problem of determining these hyperplanes is set up with the notion of *margin*. Intuitively, a *maximum margin hyperplane* is one that cleanly separates the two classes, and for which a large region (or *margin*) exists on each side of the boundary with no training data points in it. To understand this concept, the very special case where the data is *linearly separable* will be discussed first. In linearly separable data, it is possible to construct a linear hyperplane which cleanly separates data points belonging to the two classes. Of course, this special case is relatively unusual because real data is rarely fully separable, and at least a few data points, such as mislabeled data points or outliers, will violate linear separability. Nevertheless, the linearly separable formulation is crucial in understanding the important principle of maximum margin. After discussing the linear separable case, the modifications to the formulation required to enable more general (and realistic) scenarios will be addressed.

A method for the classification of both linear and nonlinear data. It uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane (i.e., a "decision boundary" separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using *support vectors* ("essential" training tuples) and *margins* (defined by the support vectors). Although the training time of even the fastest SVMs can be extremely slow, they are highly accurate, owing to their ability to model complex nonlinear decision boundaries. They are much less prone to overfitting than other methods. The support vectors found also provide a compact description of the learned model. SVMs can be used for numeric prediction as well as classification.

The Case When the Data Are Linearly Separable

To explain the mystery of SVMs, let's first look at the simplest case—a two-class problem where the classes are linearly separable. Let the data set D be given as $(\boldsymbol{X}_1, y_1)$, $(\boldsymbol{X}_2, y_2)$, … , $(\boldsymbol{X}_D, y_D)$, where $\boldsymbol{X}_i$ is the set of training tuples with associated class labels, y_i. Each y_i can take one of two values, either +1 or -1 (i.e. $y_i \in \{+1, -1\}$), corresponding to the classes *buys_ computer = yes* and *buys_computer = no*, respectively. To aid in visualization, let's consider an example based on two input attributes, A_1 and A_2, as shown in Figure below:

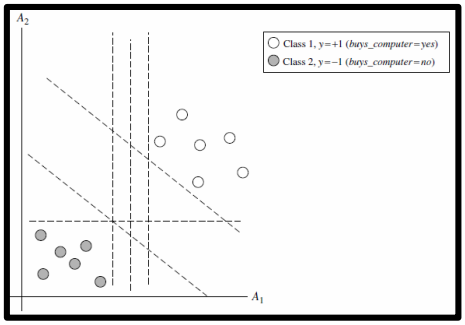


Figure: The 2-D training data are linearly separable. There are an infinite number of possible separating hyperplanes or "decision boundaries," some of which are shown here as dashed lines. Which one is best?

From the graph, we see that the 2-D data are **linearly separable** (or "linear," for short), because a straight line can be drawn to separate all the tuples of class +1 from all the tuples of class -1. There are an infinite number of separating lines that could be drawn. We want to find the "best" one, that is, one that (we hope) will have the minimum classification error on previously unseen tuples. How can we find this best line? Note that if our data were 3-D (i.e., with three attributes), we would want to find the best separating *plane*. Generalizing to *n* dimensions, we want to find the best *hyperplane*. We will use "hyperplane" to refer to the decision boundary that we are seeking, regardless of the number of input attributes. So, in other words, how can we find the best hyperplane? An SVM approaches this problem by searching for the **Maximum Marginal Hyperplane**. Consider Figure below, which shows two possible separating hyperplanes and their associated margins.

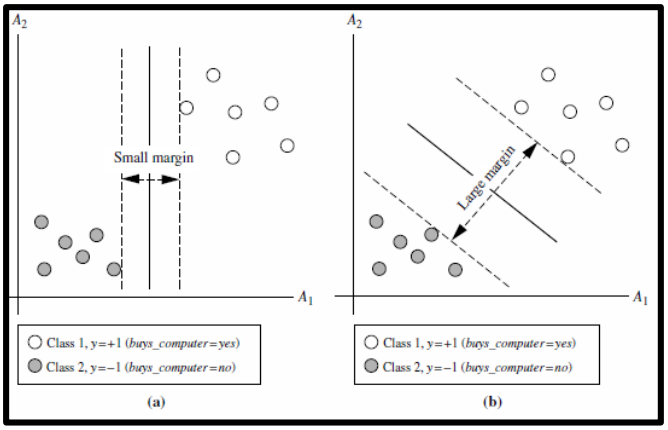


Figure: Here we see just two possible separating hyperplanes and their associated margins. Which one is better? The one with the larger margin (b) should have greater generalization accuracy.

Define the hyperplanes H such that:

$w \bullet x_i + b \geq +1$ when $y_i = +1$

$w \bullet x_i + b \leq -1$ when $y_i = -1$

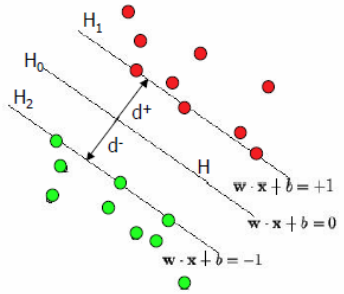
H_1 and H_2 are the planes:

H_1: $w \bullet x_i + b = +1$

H_2: $w \bullet x_i + b = -1$

The points on the planes H_1 and H_2 are the tips of the Support Vectors

The plane H_0 is the median in between, where $w \bullet x_i + b = 0$



d+ = the shortest distance to the closest positive point

d- = the shortest distance to the closest negative point

The margin (gutter) of a separating hyperplane is d+ + d–.

Before we get into the definition of margins, let's take an intuitive look at this figure. Both hyperplanes can correctly classify all the given data tuples. Intuitively, however, we expect the hyperplane with the larger margin to be more accurate at classifying future data tuples than the hyperplane with the smaller margin. This is why (during the learning or training phase) the SVM searches for the hyperplane with the largest margin, that is, the *maximum marginal hyperplane* (MMH). The associated margin gives the largest separation between classes. Getting to an informal definition of **margin**, we can say that the shortest distance from a hyperplane to one side of its margin is equal to the shortest distance from the hyperplane to the other side of its margin, where the "sides" of the margin are parallel to the hyperplane. When dealing with the MMH, this distance is, in fact, the shortest distance from the MMH to the closest training tuple of either class.

A **separating hyperplane** can be written as:

$$\mathbf{W}.\mathbf{X} + b = 0$$

Where **W** is a weight vector, namely, **W** = {w1, w2, …, wn}; n is the number of attributes; and b is a scalar, often referred to as a bias. To aid in visualization, let's consider two input attributes, A_1 and A_2, as in Figure above. Training tuples are 2-D (e.g., **X** = (x_1, x_2)), where x1 and x2 are the values of attributes A_1 and A_2, respectively, for **X**.

If we think of b as an additional weight, w_0, we can rewrite the above equation as:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

Thus, any point that lies above the separating hyperplane satisfies:

$$w_0 + w_1 x_1 + w_2 x_2 > 0$$

Similarly, any point that lies below the separating hyperplane satisfies:

$$w_0 + w_1 x_1 + w_2 x_2 < 0$$

The weights can be adjusted so that the hyperplanes defining the "sides" of the margin can be written as:

$$H_1 : w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1,$$
$$H_2 : w_0 + w_1 x_1 + w_2 x_2 \leq -1 \quad \text{for } y_i = -1.$$

That is, any tuple that falls on or above $H1$ belongs to class +1, and any tuple that falls on or below $H2$ belongs to class -1. Combining the above two inequalities:

$$y_i(w_0 + w_1 x_1 + w_2 x_2) \geq 1, \ \forall i.$$

Any training tuples that fall on hyperplanes *H*1 or *H*2 (i.e., the "sides" defining the margin) satisfy Eq. (9.18) and are called **support vectors**. That is, they are equally close to the (separating) MMH. In Figure below, the support vectors are shown encircled with a thicker border. Essentially, the support vectors are the most difficult tuples to classify and give the most information regarding classification.

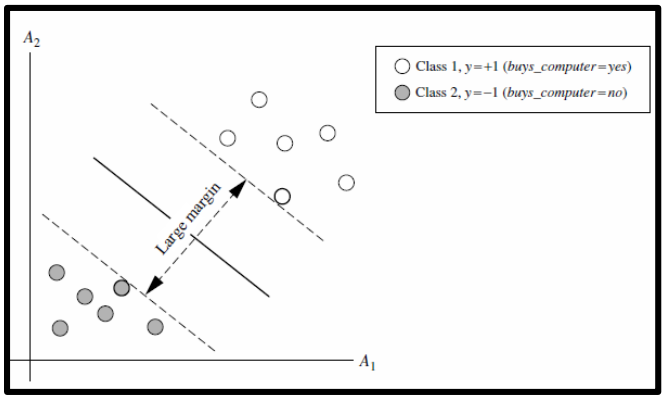


Figure: Support vectors. The SVM finds the maximum separating hyperplane, that is, the one with maximum distance between the nearest training tuples. The support vectors are shown with a thicker border.

Once we've found the support vectors and MMH (note that the support vectors define the MMH!), we have a trained support vector machine. The MMH is a linear class boundary, and so the corresponding SVM can be used to classify linearly separable data. We refer to such a trained SVM as a **Linear SVM**. The distance between the two hyperplanes for the positive and negative instances is also referred to as the margin. The goal is to maximize this margin. What is the distance (or margin) between these two parallel hyperplanes?

One can use linear algebra to show that the distance between two parallel hyperplanes is the normalized difference between their constant terms, where the normalization factor is the *L*2-norm:

$$||\overline{W}|| = \sqrt{\sum_{i=1}^{d} w_i^2}$$

of the coefficients.

Because the difference between the constant terms of the two aforementioned hyperplanes is 2, it follows that the distance between them is 2/||*W*||. This is the margin that needs to be maximized with respect to the aforementioned constraints. Note that each training data point leads to a constraint, which tends to make the optimization problem rather large, and explains the high computational complexity of SVMs. However, perfect linear separability is a rather contrived scenario, and real data sets usually will not satisfy this property. "*Once I've got a trained support vector machine, how do I use it to classify test (i.e., new) tuples?*" Based on the Lagrangian formulation mentioned before, the MMH can be rewritten as the decision boundary.

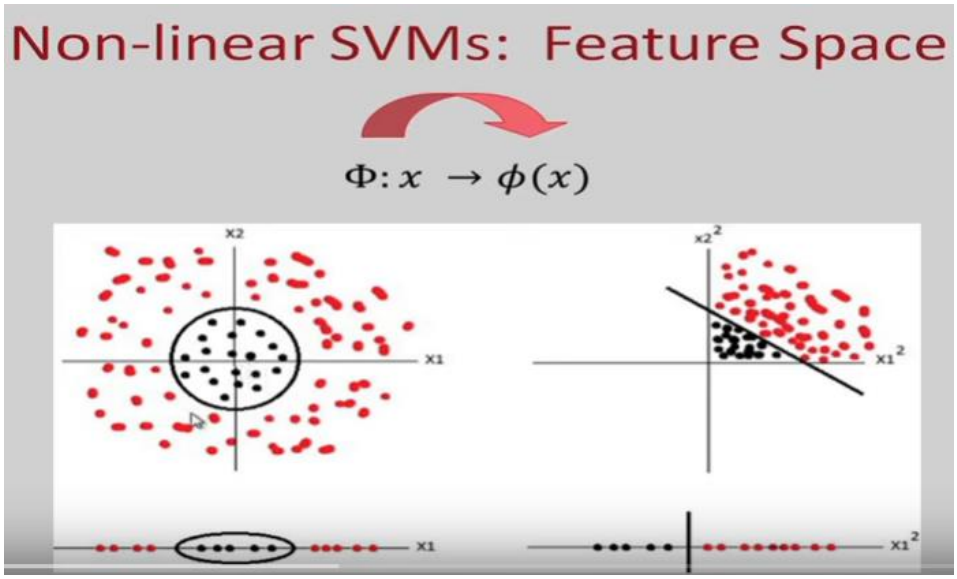
$$d(\boldsymbol{X}^T) = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{X}_i \boldsymbol{X}^T + b_0$$

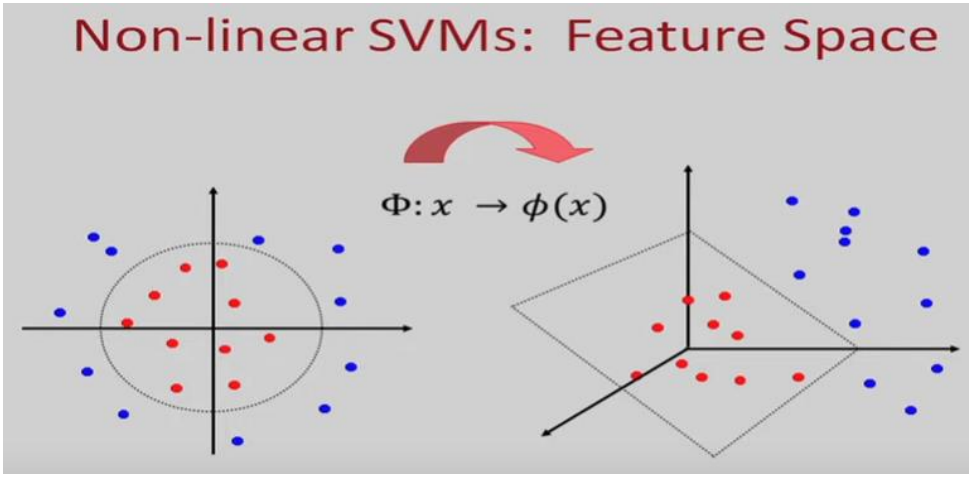
where y_i is the class label of support vector $\boldsymbol{X}_i$; $\boldsymbol{X}^T$ is a test tuple; α_i and b_0 are numeric parameters that were determined automatically by the optimization or SVM algorithm noted before; and l is the number of support vectors.

α_i are Lagrangian multipliers. For linearly separable data, the support vectors are a subset of the actual training tuples. Given a test tuple, $\boldsymbol{X}^T$, we plug it into the above equation, and then check to see the sign of the result. ***This tells us on which side of the hyperplane the test tuple falls.*** If the sign is positive, then $\boldsymbol{X}^T$ falls on or above the MMH, and so the SVM predicts that $\boldsymbol{X}^T$ belongs to class +1 (representing *buys computer* D *yes*, in our case). If the sign is negative, then $\boldsymbol{X}^T$ falls on or below the MMH and the class prediction is -1 (representing *buys computer* D *no*).

Notice that the Lagrangian formulation of our problem contains a dot product between support vector $\boldsymbol{X}_i$ and test tuple $\boldsymbol{X}^T$. This will prove very useful for finding the MMH and support vectors for the case when the given data are nonlinearly separable. Before we move on to the nonlinear case, there are two more important things to note. The complexity of the learned classifier is characterized by the number of support vectors rather than the dimensionality of the data. Hence, SVMs tend to be less prone to overfitting than some other methods. The support vectors are the essential or critical training tuples—they lie closest to the decision boundary (MMH). If all other training tuples were removed and training were repeated, the same separating hyperplane would be found. Furthermore, the number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality. An SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high.

SVM – KERNEL FUNCTION (Prof. Sudheshna Sarkar)





The kernel functions play a very important role in SVM. Their job is to take data as input and they transform it in any required form. They are significant in SVM as they help in determining various important things.

In this article, we will be looking at various types of kernels. We will also be looking at how the kernel works and why is it necessary to have a kernel function.

This will be an important article, as it will give an idea of what kernel function should be used in specific programs. We have already discussed about ***SVM in Machine Learning*** and ***SVM applications*** in the previous articles. If you haven't read those articles, first read them.

What is Kernel?

A kernel is a function used in SVM for helping to solve problems. They provide shortcuts to avoid complex calculations. The amazing thing about kernel is that we can go to higher dimensions and perform smooth calculations with the help of it. We can go up to an infinite number of dimensions using kernels.

Sometimes, we cannot have a hyperplane for certain problems. This problem arises when we go up to higher dimensions and try to form a hyperplane. A kernel helps to form the hyperplane in the higher dimension without raising the complexity.

Working of Kernel Functions

Kernels are a way to solve non-linear problems with the help of linear classifiers. This is known as the **kernel trick** method. The kernel functions are used as parameters in the SVM codes. They help to determine the shape of the hyperplane and decision boundary. We can set the value of the kernel parameter in the SVM code. The value can be any type of kernel from linear to polynomial. If the value of the kernel is linear then the decision boundary would be linear and two-dimensional.

These kernel functions also help in giving decision boundaries for higher dimensions. We do not need to do complex calculations. The kernel functions do all the hard work. We just have to give the input and use the appropriate kernel. Also, we can solve the overfitting problem in SVM using the kernel functions.

Overfitting happens when there are more feature sets than sample sets in the data. We can solve the problem by either increasing the data or by choosing the right kernel. There are kernels like RBF that work well with smaller data as well. But, RBF is a universal kernel and using it on smaller datasets might increase the chances of overfitting. Hence, the use of simpler kernels like linear and polynomial is encouraged.

Kernel

- Original input attributes is mapped to a new set of input features via feature mapping Φ.
- Since the algorithm can be written in terms of the scalar product, we replace $x_a . x_b$ with $\phi(x_a).\phi(x_b)$
- For certain Φ's there is a simple operation on two vectors in the low-dim space that can be used to compute the scalar product of their two images in the high-dim space

$$K(x_a, x_b) = \phi(x_a).\phi(x_b)$$

Let the kernel do the work rather than do the scalar product in the high dimensional space.

Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b = \sum_{i \in SV} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

- We only use the dot product of feature vectors in both the training and test.
- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(x_a, x_b) = \phi(x_a).\phi(x_b)$$

Kernel Example

2-dimensional vectors $\overline{x} = [x_1 x_2]$

let $K(x_i, x_j) = (1 + x_i . x_j)^2$

We need to show that $K(x_i, x_j) = \phi(x_i).\phi(x_j)$

$K(x_i,x_j) = (1 + x_i x_j)^2$,

$= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$

$= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1} x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2} x_{i1} \;\; \sqrt{2} x_{i2}] . [1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1} x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2} x_{j1} \;\; \sqrt{2} x_{j2}]$

$= \phi(x_i).\phi(x_j)$,

where $\phi(x) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1 x_2 \;\; x_2^2 \;\; \sqrt{2} x_1 \;\; \sqrt{2} x_2]$

Commonly-used kernel functions

- Linear kernel: $K(x_i, x_j) = x_i . x_j$
- Polynomial of power p:

$$K(x_i, x_j) = (1 + x_i . x_j)^p$$

- Gaussian (radial-basis function):

$$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

- Sigmoid

$$K(x_i, x_j) = \tanh(\beta_0 x_i . x_j + \beta_1)$$

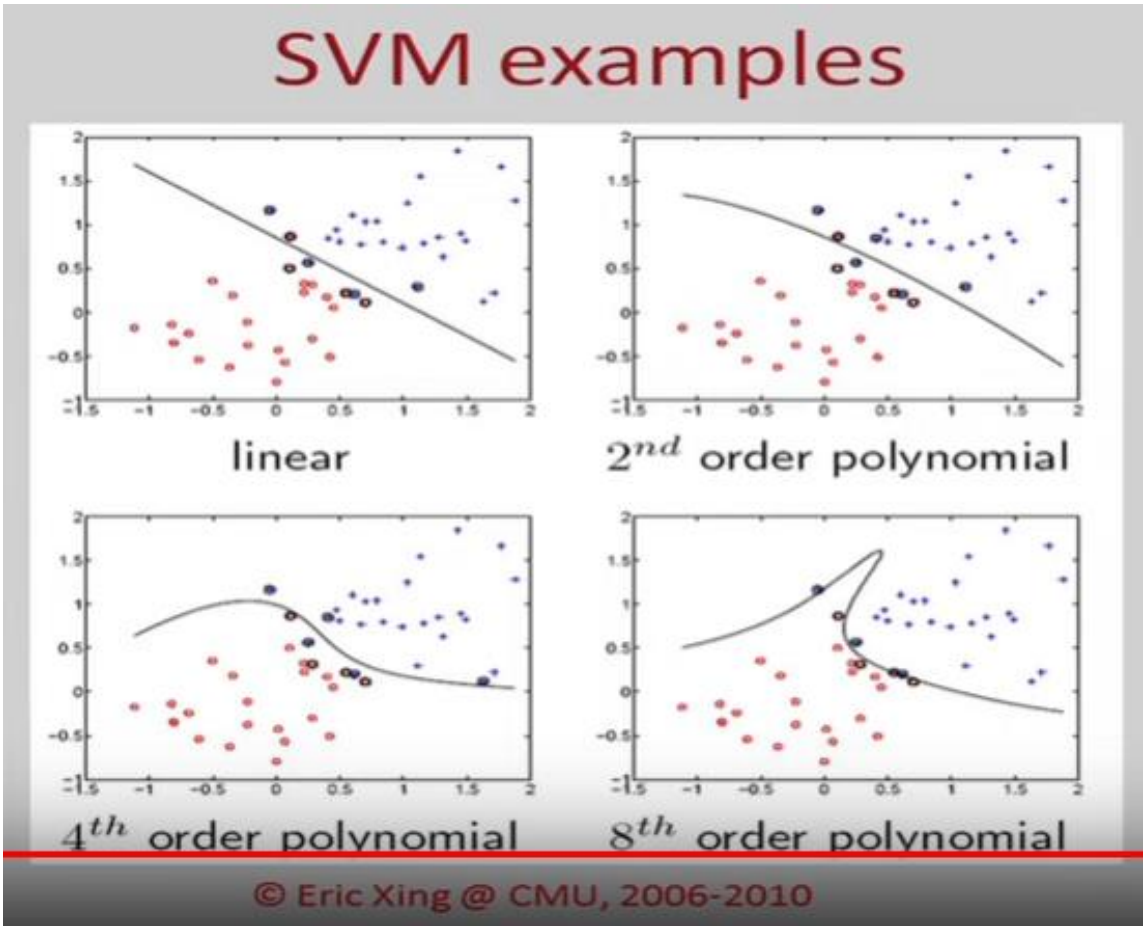
In general, functions that satisfy *Mercer's condition* can be kernel functions.

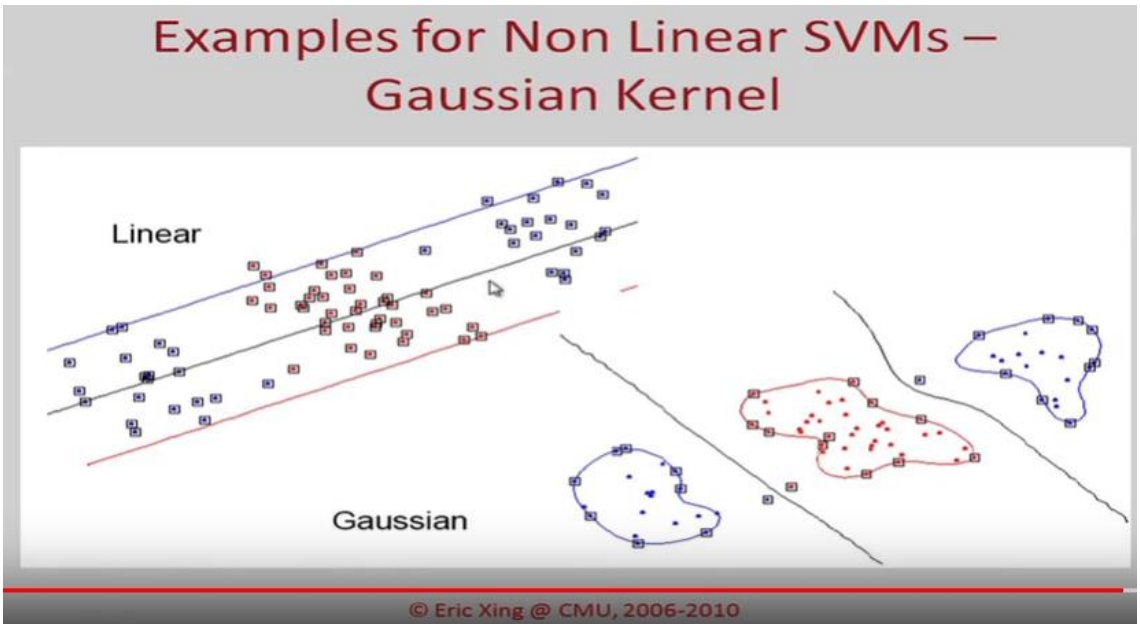
Kernel Functions

- Kernel function can be thought of as a similarity measure between the input objects
- Not all similarity measure can be used as kernel function.
- Mercer's condition states that any positive semi-definite kernel $K(x, y)$, i.e.

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

- can be expressed as a dot product in a high dimensional space.


SVM examples
linear
2^{nd} order polynomial
4^{th} order polynomial
8^{th} order polynomial
© Eric Xing @ CMU, 2006-2010


Examples for Non Linear SVMs – Gaussian Kernel
Linear
Gaussian
© Eric Xing @ CMU, 2006-2010

Nonlinear SVM: Optimization

- Formulation: (Lagrangian Dual Problem)

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{such that} \quad 0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- The solution of the discriminant function is

$$g(x) = \sum_{i \epsilon SV} \alpha_i K(x_i, x_j) + b$$

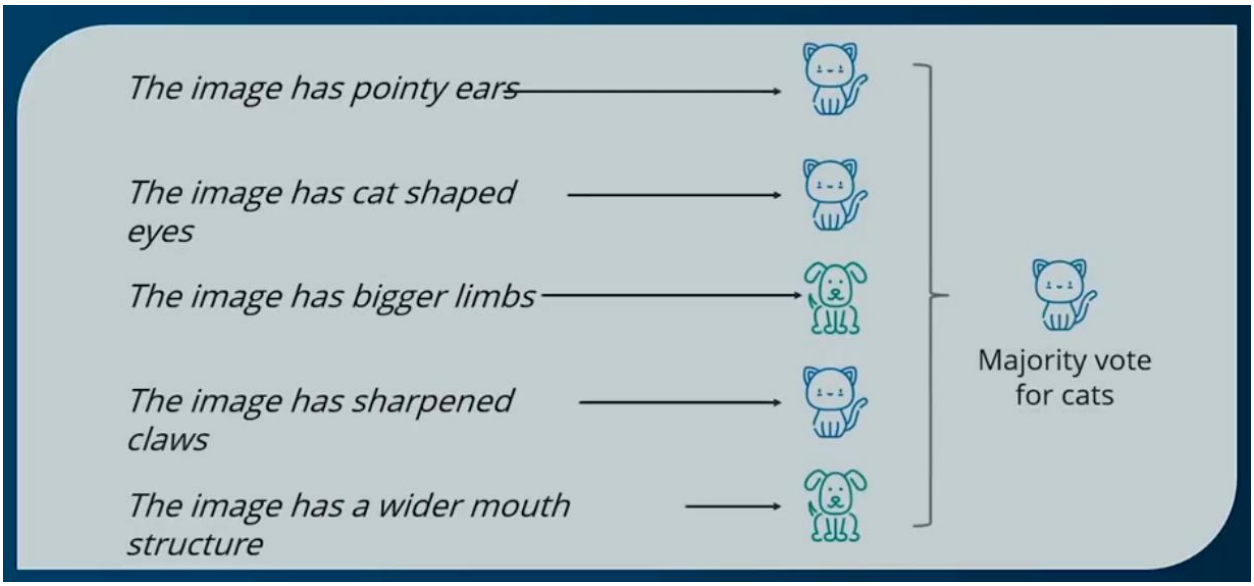
Performance

- Support Vector Machines work very well in practice.
  - The user must choose the kernel function and its parameters
- They can be expensive in time and space for big datasets
  - The computation of the maximum-margin hyper-plane depends on the *square of the number of training cases.*
  - We need to store all the support vectors.
- The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space.

WHY IS BOOSTING USED?
WHAT IS BOOSTING IN MACHINE LEARNING?
HOW DOES BOOSTING ALGORITHM WORK?
TYPES OF BOOSTING

01

WHY IS BOOSTING USED?

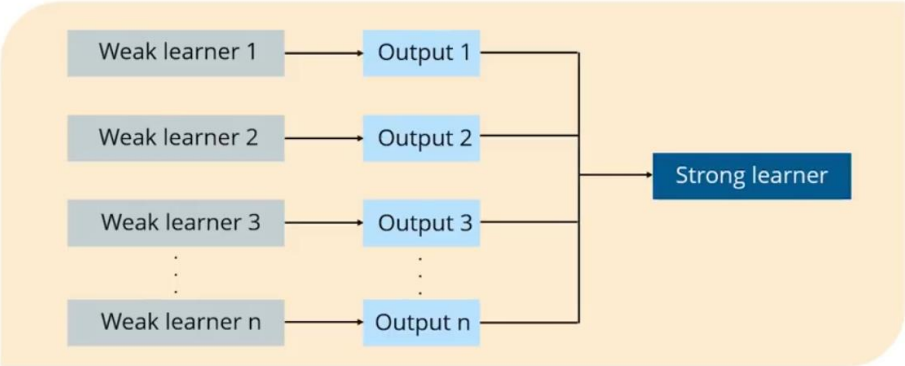


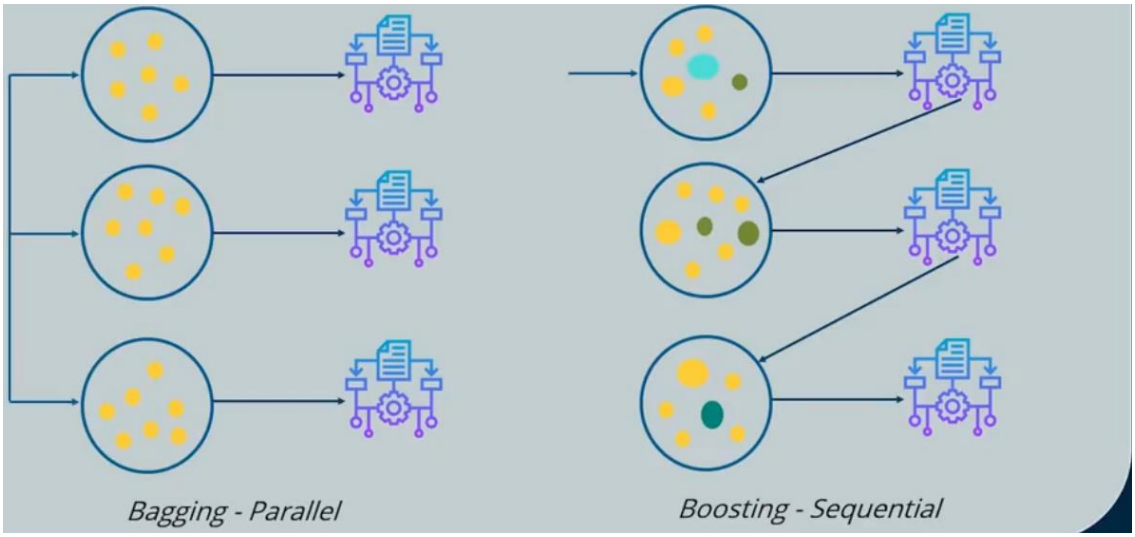
Why Is Boosting Used?

These rules help us identify whether an image is a Dog or a cat but, if we were to classify an image based on an individual (single) rule, the prediction would be flawed. Each of these rules, individually, are called weak learners because these rules are not strong enough to make predictions.

WHAT IS BOOSTING?

Boosting is a process that uses a set of Machine Learning algorithms to combine weak learner to form strong learners in order to increase the accuracy of the model.

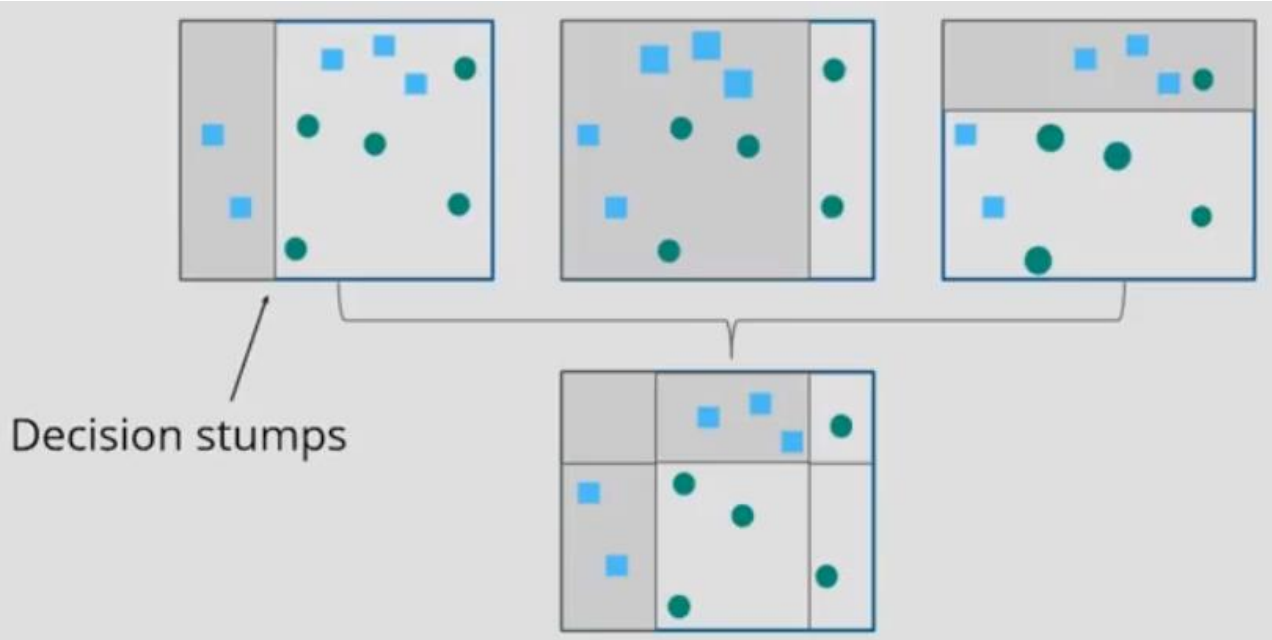




What Is Ensemble Learning?

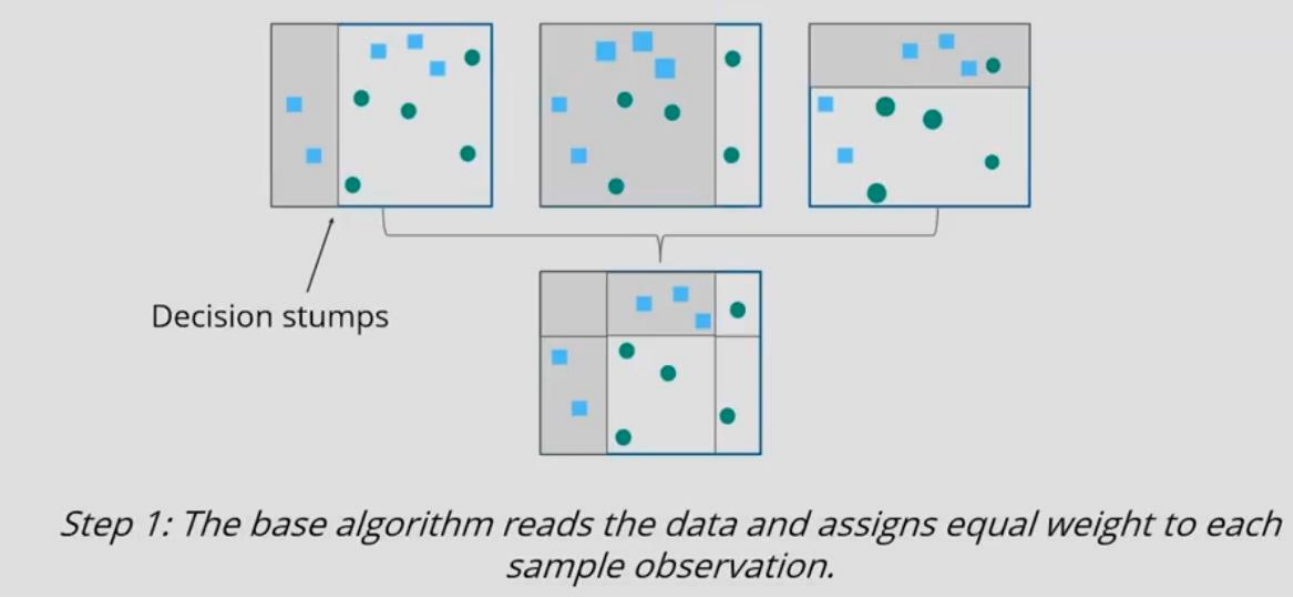
Ensemble learning is a method that is used to enhance the performance of Machine Learning model by combining several learners. When compared to a single model, this type of learning builds models with improved efficiency and accuracy

03
HOW DOES BOOSTING ALGORITHM WORK?

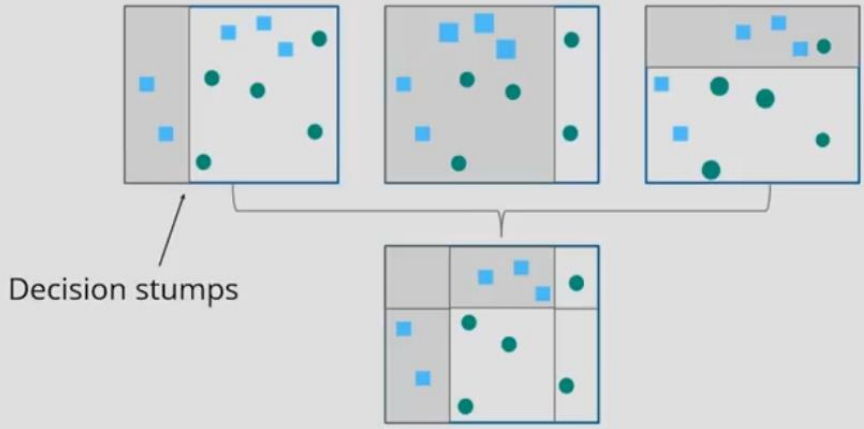


How Does Boosting Algorithm Work?

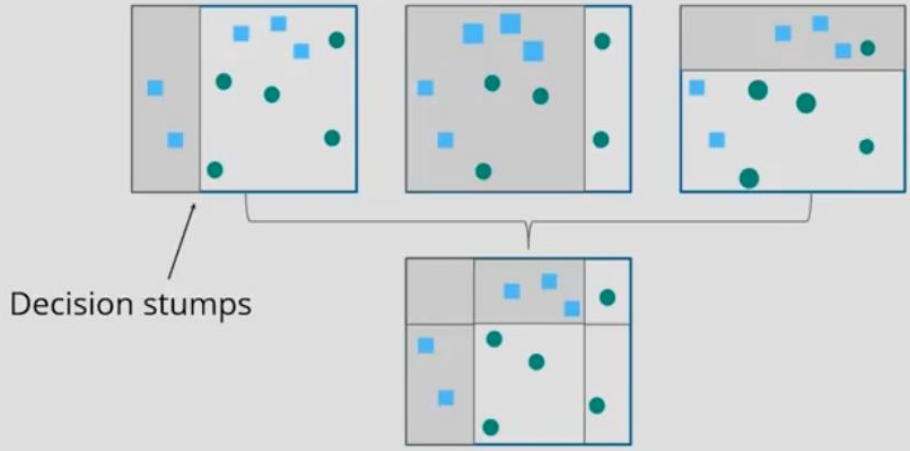
The basic principle behind the working of the boosting algorithm is to generate multiple weak learners and combine their predictions to form one strong rule



Step 1: The base algorithm reads the data and assigns equal weight to each sample observation.



Step 2: False predictions are assigned to the next base learner with a higher weightage on these incorrect predictions.

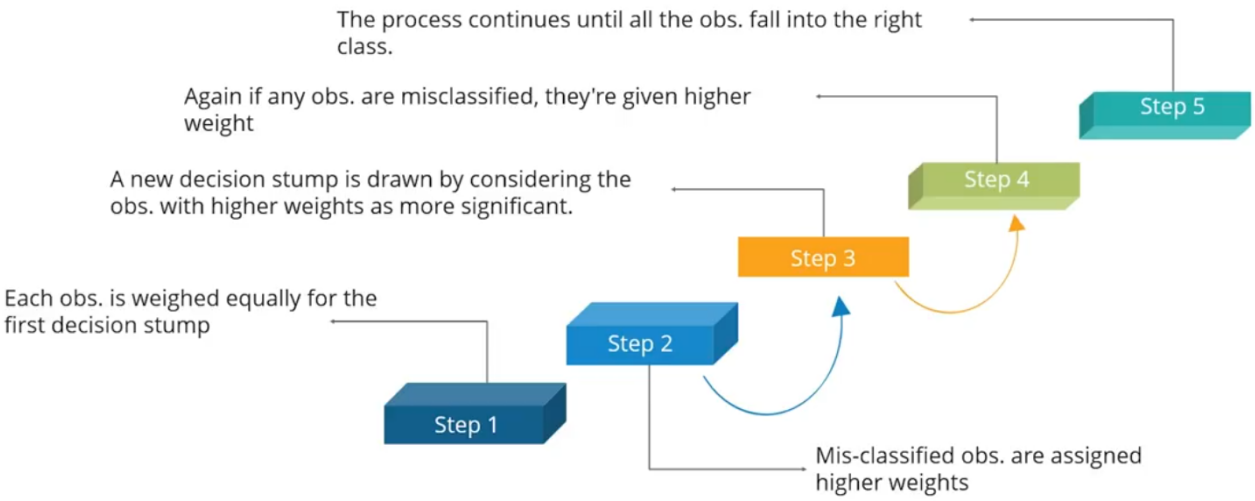


Step 3: Repeat step 2 until the algorithm can correctly classify the output.

04

TYPES OF BOOSTING

ADAPTIVE BOOSTING



GRADIENT BOOSTING

In Gradient Boosting, base learners are generated sequentially in such a way that the present base learner is always more effective than the previous one.

optimize the loss function of the previous learner

Three main components:

- ***Loss function*** *that needs to be ameliorated.*
- ***Weak learner*** *for computing predictions and forming strong learners.*
- *An* ***Additive Model*** *that will regularize the loss function.*

XGBOOST

XGBoost is an advanced version of Gradient boosting method that is designed to focus on computational speed and model efficiency.