An Introduction to ML Algorithms

by Nawwaf Kharma, PhD (DIC)

Advanced Organizer

This lecture introduces you to:

Regression

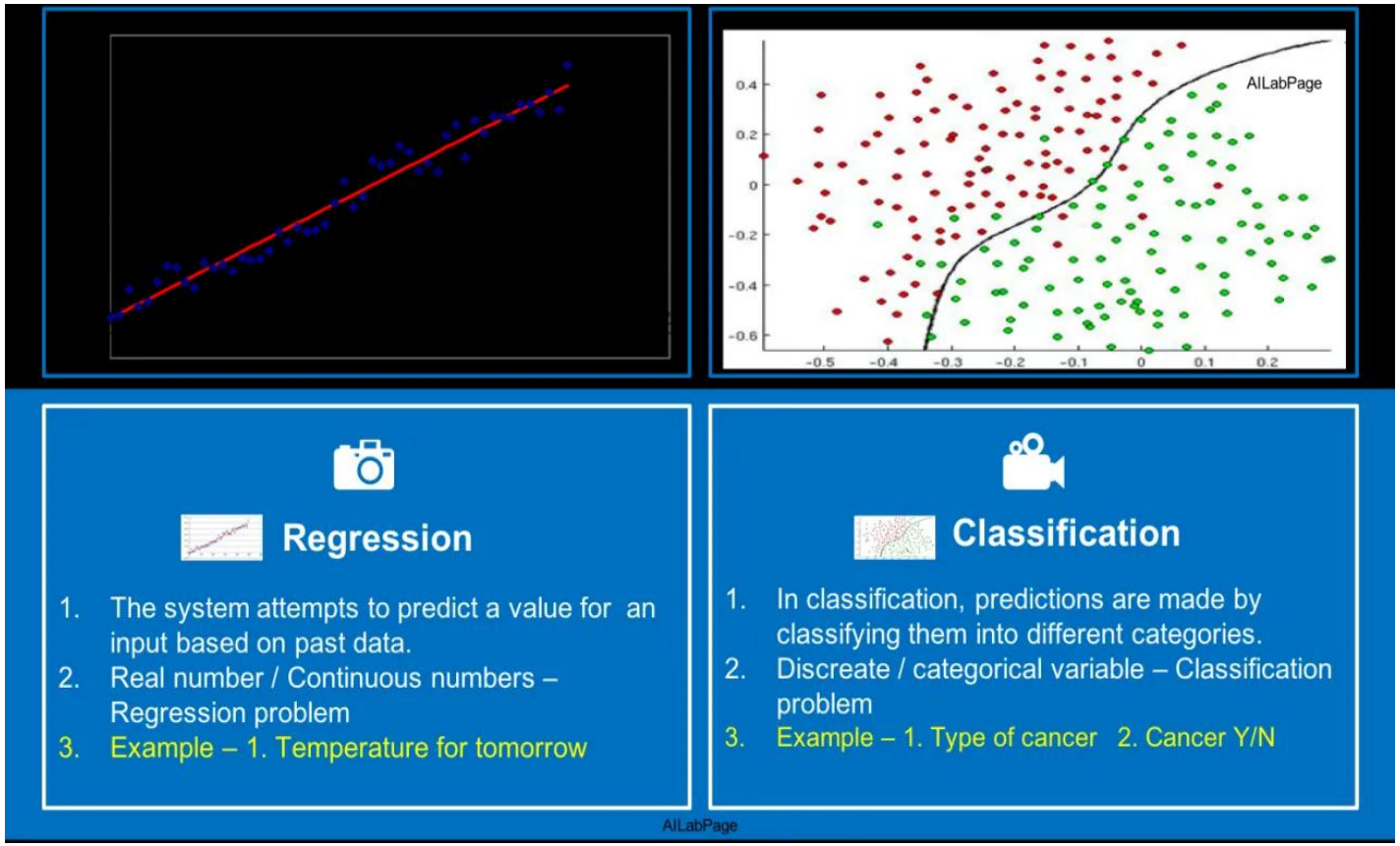
- 1- and N-input Linear Regression
- Quadratic Regression
- Potential Problems

Classification

- Logistic Regression
- Tree-based Classification
- Support Vector Classification

Clustering

- K-means Clustering



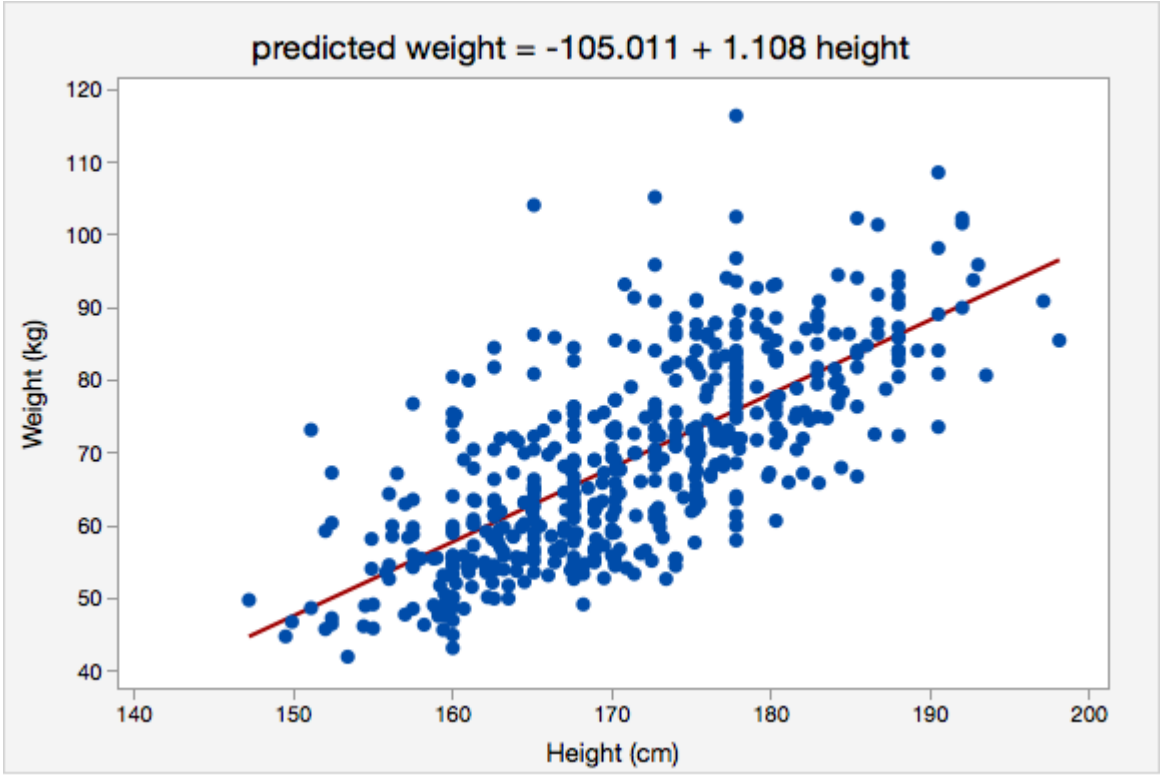
Linear Regression

- Linear regression is about fitting a line to a set of (say n= 100) input or training instances (x_i, y_i), where x is called a *predictor* variable (feature) and y is the *response* variable (class or value);

- The fitted line will have the form: **y** (x) = b_0 + b_1*x where **y** is the predicted response, b_0 is the (y) *intercept* of the regression line and b_1 is the *slope* of the line;

- Once the *coefficients* (b_0 and b_1) are found, the formula can be used to *predict*, for example, the weight of a person, given his/her height;

- Most predictions will likely differ from reality; the difference between *actual* and values (when assessed using instances NOT used in training) is called *test error.*



Coefficients b_0 = -105.011 and b_1=1.108

Estimating the Coefficients

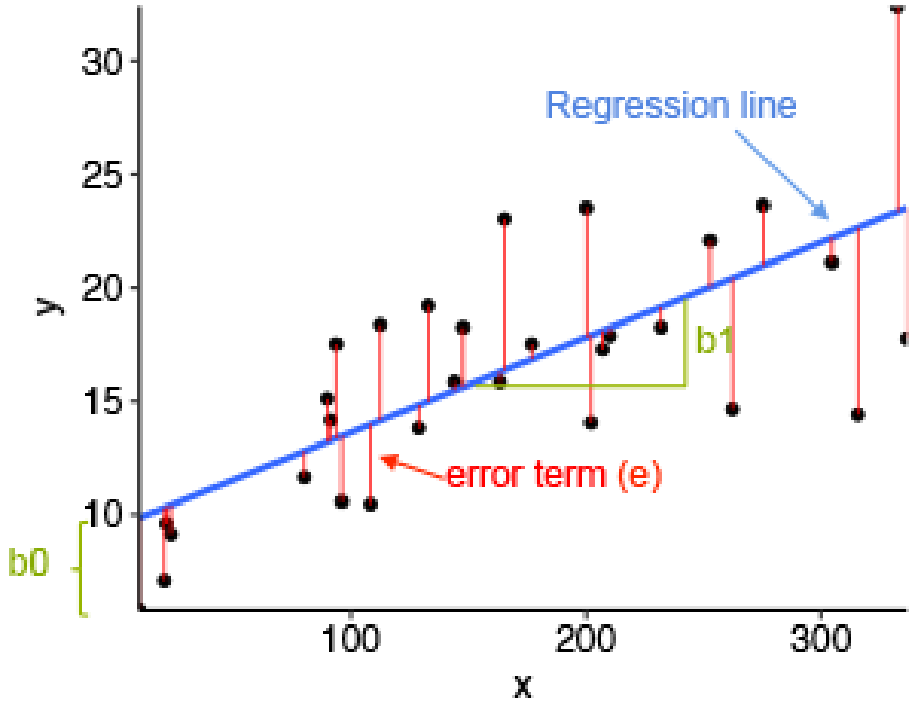
- When estimating the coefficients, b_0 and b_1, the objective is to minimize RSS, which is the sum of the squares of the individual error terms (or *residuals*):

$$RSS = \sum_{i=1}^{n}(y_i - (b_0 + b_1 * x_i))^2$$

- RSS is **minimized by setting the following values** for the coefficients

$$b_1 = \sum_{i=1}^{n}\left((x_i - \boldsymbol{mean}(x)) * \left(y_i - \boldsymbol{mean}(y)\right)\right) / \sum_{i=1}^{n}(x_i - \boldsymbol{mean}(x))^2$$

$$b_0 = \boldsymbol{mean}(y) - b_1 * \boldsymbol{mean}(x)$$



Assessing the Accuracy of Coefficient Estimates

- If you had the whole *population* of instances then the best fit is called a population regression line. In most cases, the line fit is a *sample* regression line;

- The population regression line is the best possible regression line: the diagram opposite shows multiple regression lines resulting from different samples (of the population);

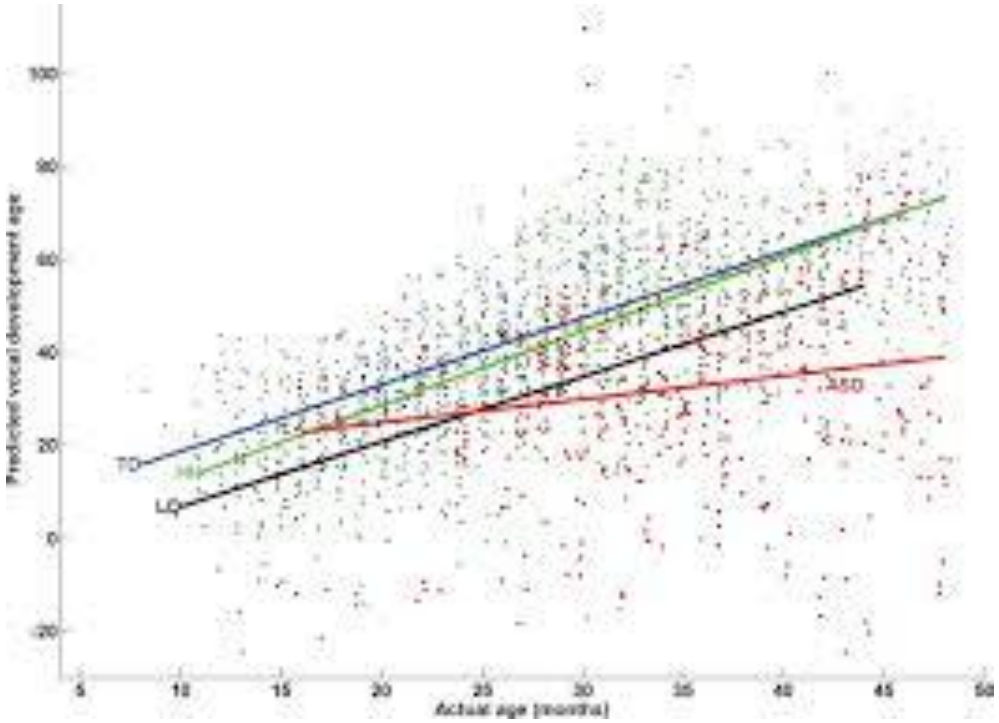
- Naturally, the questions that follow are:

Q1. How can we train the model in a manner that generates a regression line closest to the optimal population regression line?

A1. ***enlarge*** the sample A2: ***re-sample and average*** the found coefficients

Q2. How can we assess the difference between a regression line and its optimal target?

A2. Use confidence intervals (next slide)



Confidence Intervals

A 95% confidence interval for b_1 and b_0 is calculated using

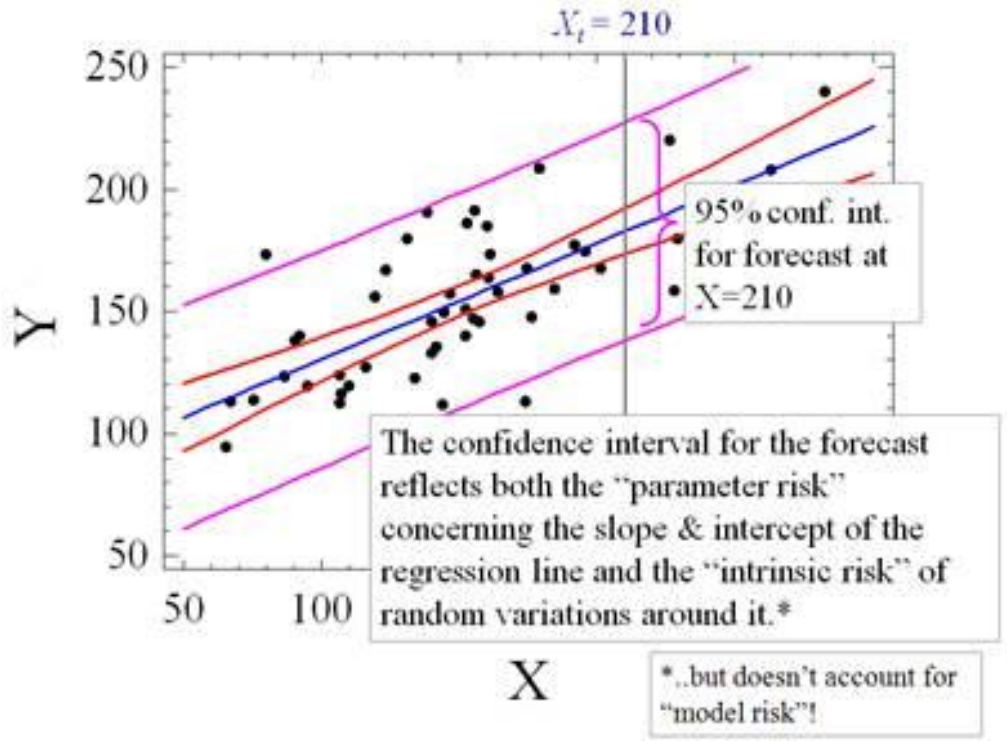
b_1 +/- 2 * SE(b_1) and b_0 +/- 2*SE(b_0), where

SE(b_1) = sqrt [SD^2 (1/n + mean(x) / $\sum_{i=1}^{n}(x_i - \boldsymbol{mean}(\boldsymbol{x}))^2$))]

SE(b_0) = sqrt [SD^2 / $\sum_{i=1}^{n}(x_i - \boldsymbol{mean}(\boldsymbol{x}))^2$], and

SD is standard deviation =~ residual standard error (RSE),

RSE = sqrt [RSS / (n – 2)] , and RSS = $\sum_{i=1}^{n}(\boldsymbol{y_i} - y_i)^2$



Accuracy of Model

These are two ways of judging the *training accuracy* of the model:

A. Residual Standard Error (RSE)

RSE = sqrt [RSS / (n – 2)] , and RSS = $\sum_{i=1}^{n}(\boldsymbol{y_i} - y_i)^2$

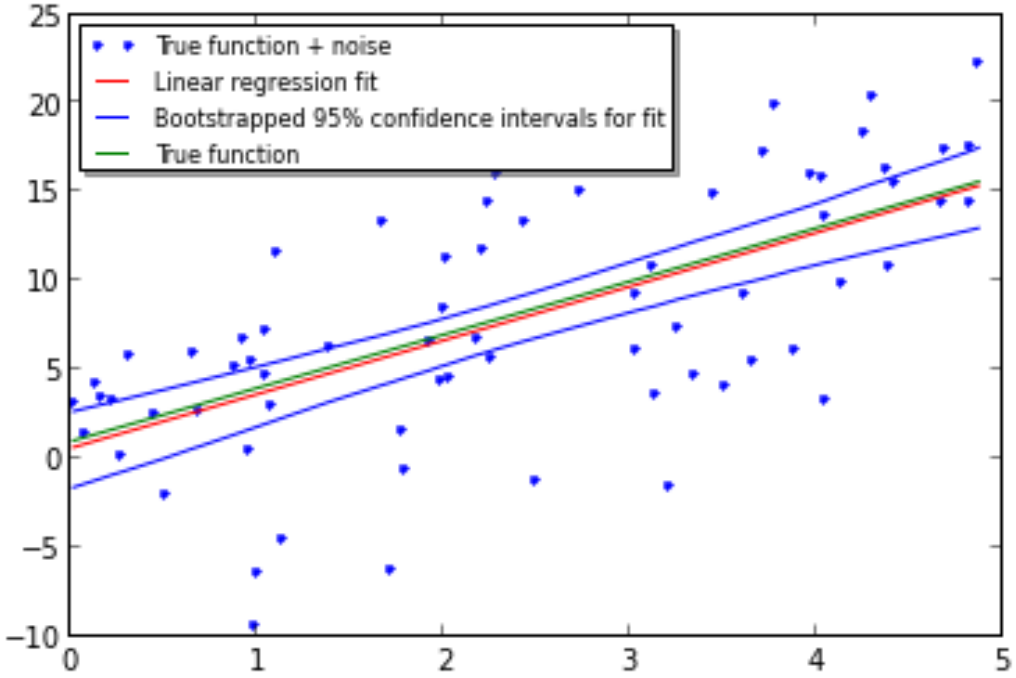
Which estimates how well the trained model fits the training data (or, deviates from the ideal population regression line).

B. **The R^2 Statistic**

R^2 = (TSS – RSS)/TSS, where TSS = $\sum_{i=1}^{n}(y_i - \text{mean}(y))^2$

R^2 is the he proportion of the total variation in the response (or TSS) that is explained by X. If the linear model fits the training data very well, RSS will be close to 0 and hence, R^2 =~1.

Note: for simple linear models *Correlation*^{2} is identical to R^2



True function line = population regression line
Linear regression fit = sample regression line
Boundaries of 95% confidence interval (blue lines)
Blue points = a sample of training instances

Multiple Linear Regression

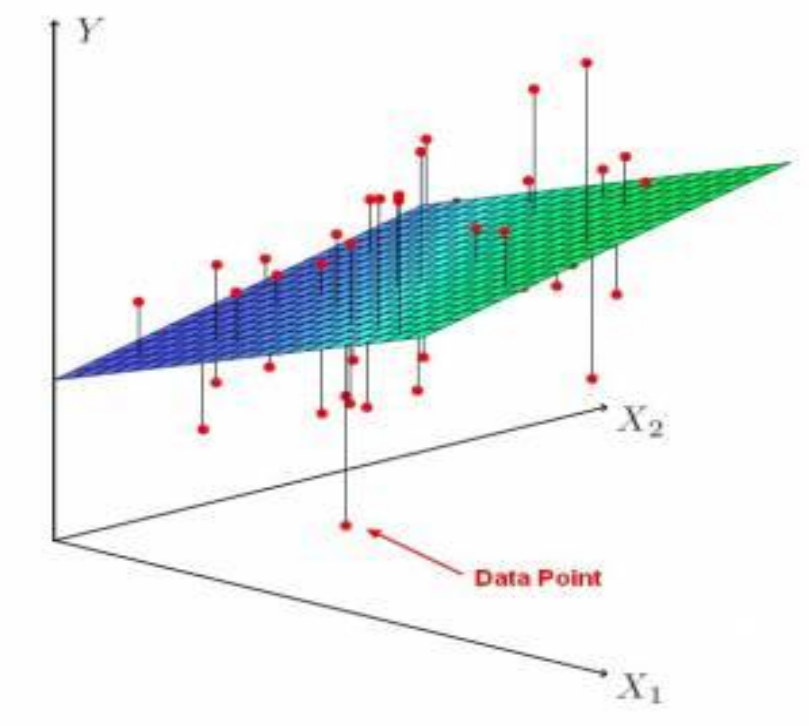
Multiple (p) predictors (x_1, .., x_p) are used to predict the single response **y**. The multiple linear regression formula has the form:

$$\mathbf{y}\,(x) = b_0 + b_1{*}x_1 + b_2{*}x_2 + .. + b_p{*}x_p$$

As in the simple regression model, the challenge is to find the *coefficients* b_i that minimize RSS, over all training instances (n), where:

$$RSS = \sum_{i=1}^{n}(\boldsymbol{y_i} - y_i)^2$$

Note: there is no reason why we could not extend the number of predictors beyond the 2 shown the figure opposite: that is just an example.



The red points are the training data;
The flat plane is the estimated response function;
The vertical lines present training errors (difference between estimate and actual values, for the training sample)

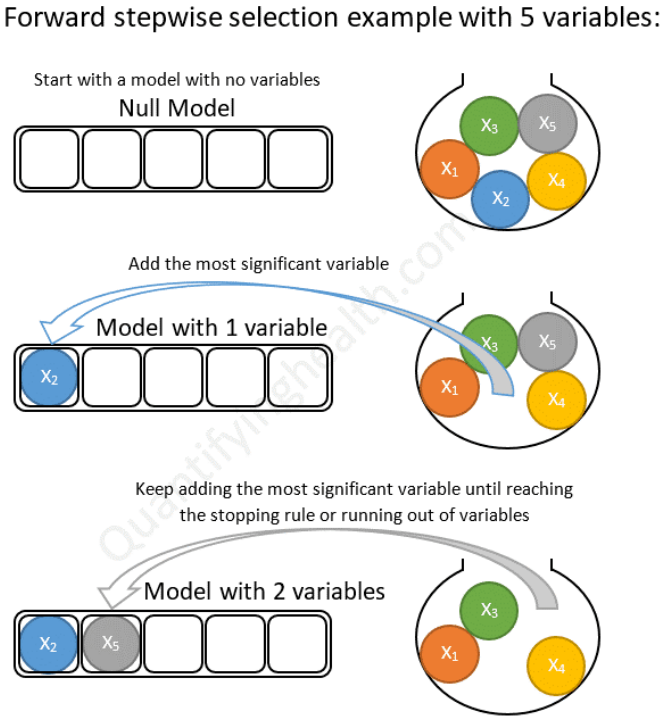
Important Considerations

A. How do we decide on a subset of variables that should be used as predictors for a given response? For v variables, there are 2^v possible subsets; selection methods include:

A1. **Forward Selection**

step 1: Let $\mathbf{y}(X) = b_0$, were b_0 is a reasonable initial estimate (***mean*** (y_i))
step 2: Extend the equation by adding the variable that gives the **lowest RSS**
step 3: If a pre-set stopping rule is satisfied (e.g., *test error* < pre-set threshold) then stop, else go to step 2

Note test error is NOT training error. Test error can be assessed via k-fold cross-validation, explained in the next slide.



Forward selection for a multi-variate model with up to 5 predictors (x_1 to x_5)

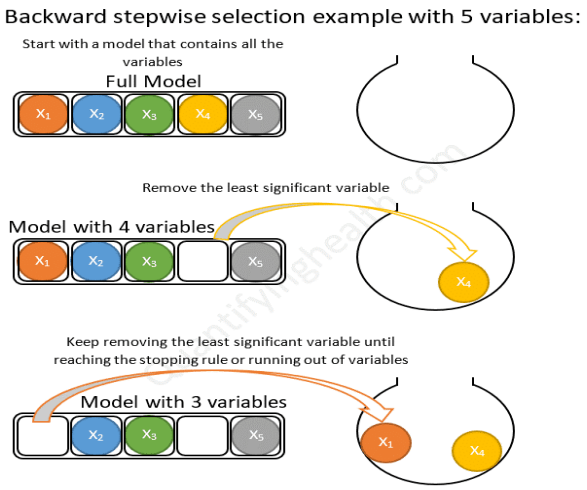
Important Considerations cont.

A2. **Backward Selection**

step 1: start with all the variables in the model;

step 2: remove the variable that least increases RSS (or the one that least significant);

Step 3: stop when a stopping rule is satisfied.



B. Model Fit

B1. **Correlation**(y, **y**)^2 , where

y is actual value while **y** is the predicted one.

B2. **RSE** = sqrt (RSS / (n – p – 1)) , where

n is number of *instances* in training data

p is number of *predictors* used

and **RSS** = $\sum_{i=1}^{n}(\boldsymbol{y_i} - y_i)^2$

k-fold Cross Validation (or CV)

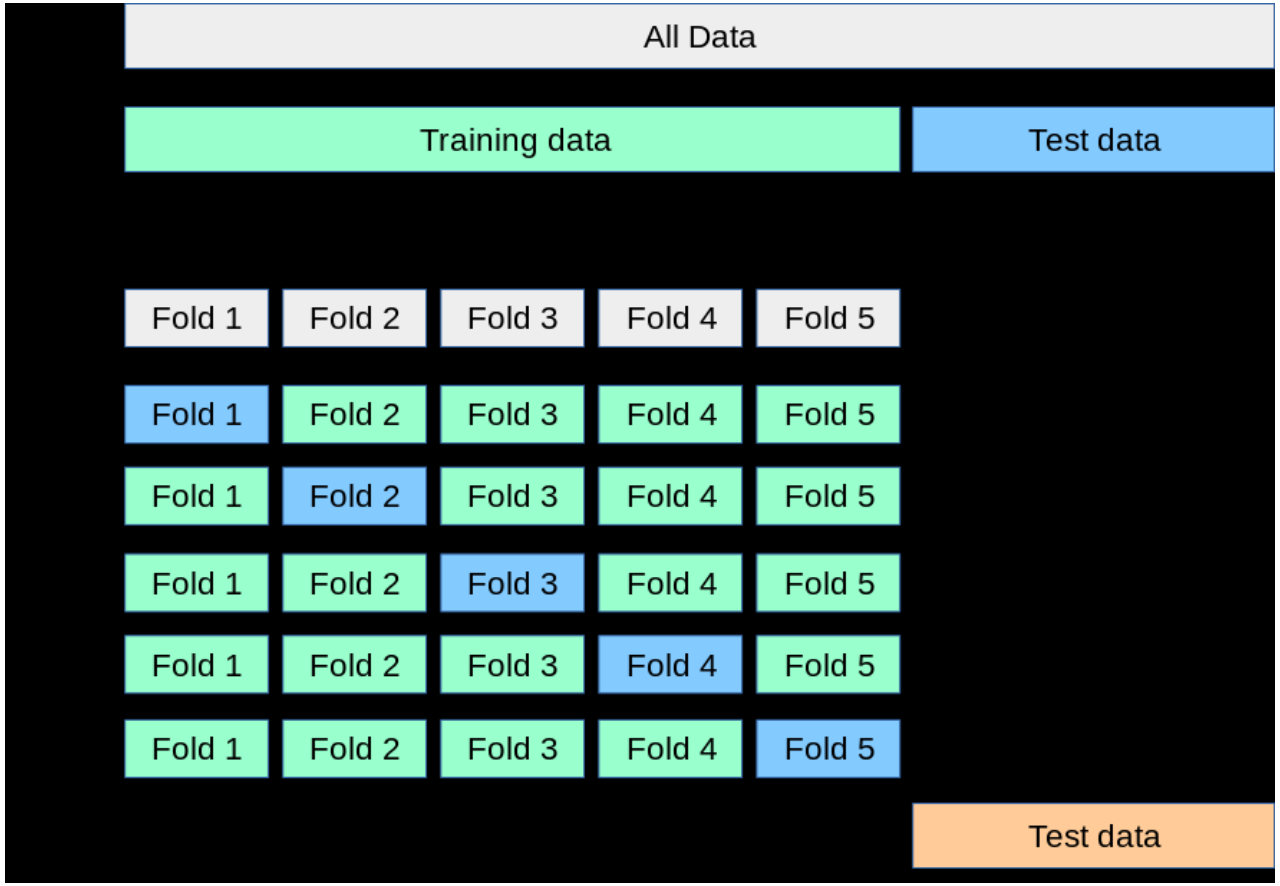
In **k-fold Cross Validation** [k here = 5, usually 10]

Step1: The training data is partitioned into k equally-sized partitions;

Step 2: The model is trained (i.e., regression line or surface computed) using the training data in one of the (k-1) folds;

Step 3: the accuracy of the resulting model (i.e., **y**=f(X) prediction equation) is assessed using the remaining fold;

Steps 2-3 are repeated k times, then the final test accuracy is computed = ***mean***(individual accuracies).



Other Considerations

A. Qualitative Predictors

Predictors with only 2 levels. Let's say we wish to predict the **age of death based on gender**- meaning when is a person likely to die if he were male or female?

In this case, the *predictor* can take only one of two values. These values have to be encoded numerically.

Say we assign x = 1 if female, and -1 if male, **y** is the predicted age of death

$$\mathbf{y}(x) = b_0 + b_1 * x$$

B. Qualitative Responses

Response with only 2 levels. Let's say we wish to predict **whether someone's were male or female**, if he/she died at a given age.

In this case, the *response* can take one of two values, while the predictor (age at death) is an integer.

x = age, say in [0, 100], while y = 1 if female and -1 if male

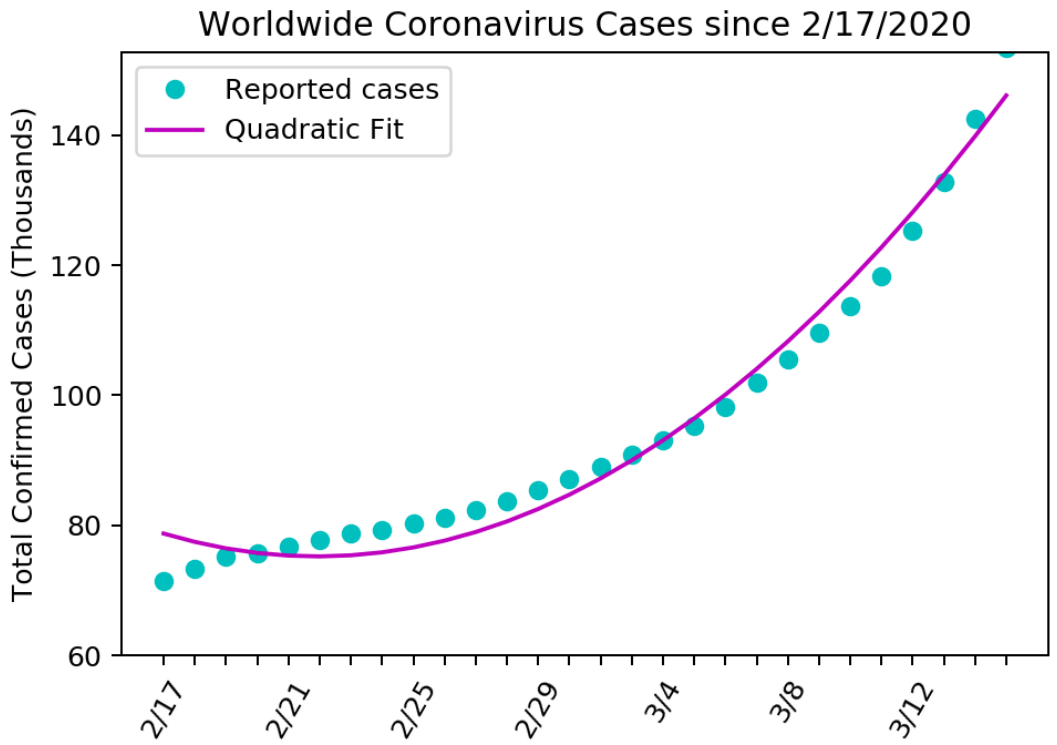
$$\mathbf{y}(x) = b_0 + b_1 * x$$

Other Considerations cont.

C. Non-linear Relationships

All previous examples presumed a linear relationship between predictors and response; that may be a false assumption, such as the case with the relationship between time and no. of cases of COVID-19 infections. For such a situation, one would use a function such as:

$$\mathbf{y}(\text{time}) = b_0 + b_1 * \text{cases} + b_2 * \text{cases}^2$$



Some Potential Problems

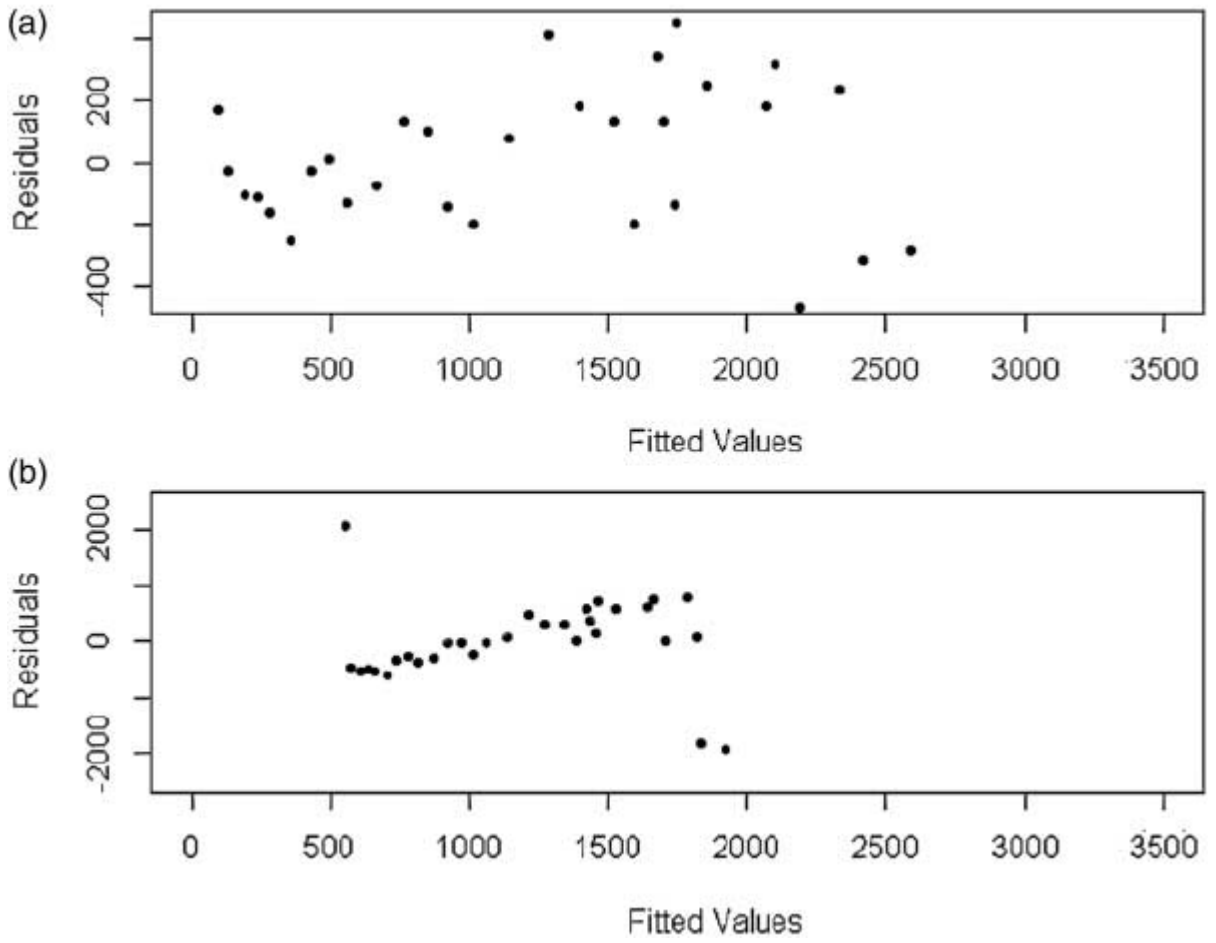
A. Outliers

Outliers are instances (points) whose actual value (y_i) is **far from their model-predicted value** ($\boldsymbol{y}_i$);

They arise for multiple reasons, including unsuitability of model and noise in means of recoding/transmitting data;

If a model fits the data extremely well, except for a couple of outliers, then it is a good idea to identify, investigate and possibly remove the outliers and hence, re-build the model;

One way to identify outliers that is to plot the fitted (i.e., predicted) values vs. the residual errors ($\boldsymbol{y}_i$-y_i). The top figure does not appear to have any potential outliers while the bottom one has three.



Some Potential Problems cont.

B. High-Leverage Points

An instance with a high (i.e., far from the mean) value for a **predictor** (x_i) **is considered a high-leverage point**;

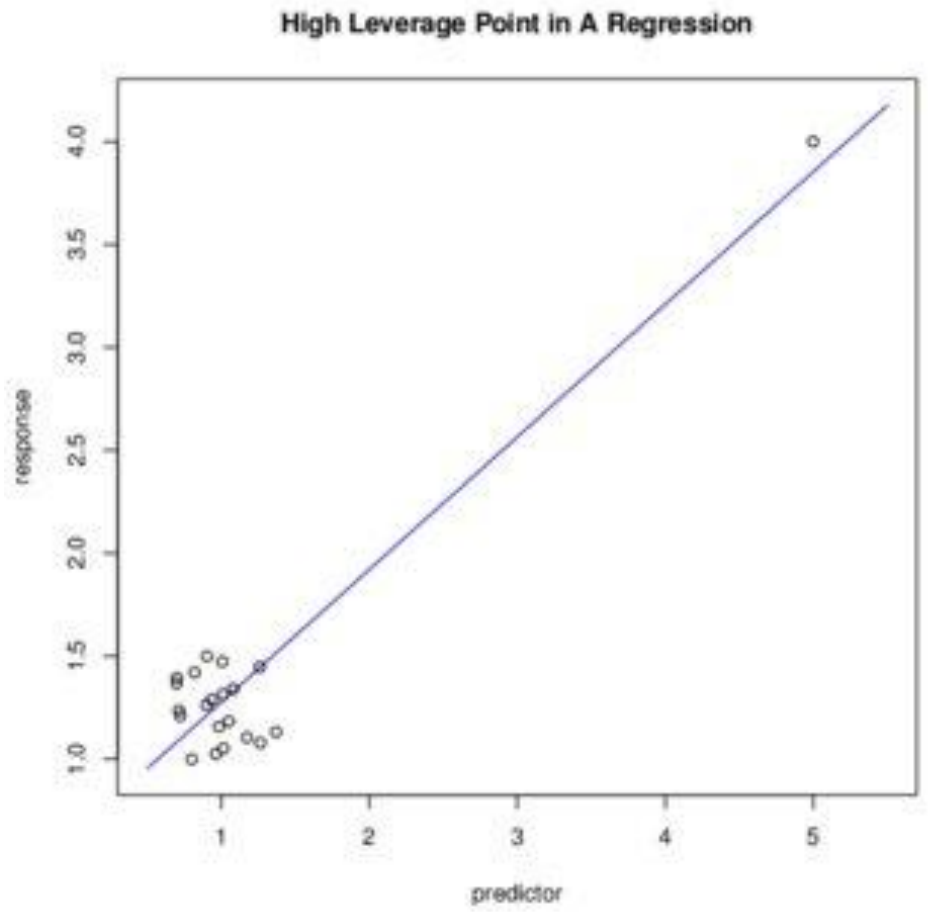
Leverage (h), for linear regression with a single-predictor is:

$$h_i = \mathbf{1/n + (x_i - mean(x))^2 / \sum_{j=1}^{n}(x_j - mean(x))^2}$$

This formula has a general form (not shown) that extends to any number of predictors;

High-leverage points result in models that, on average, fit the training data well, but result in bad test accuracies;

As such, unless they are needed, it is best that such points are also identified and removed, and the model re-trained.



Linear Vs. Logistic Regression for Classification

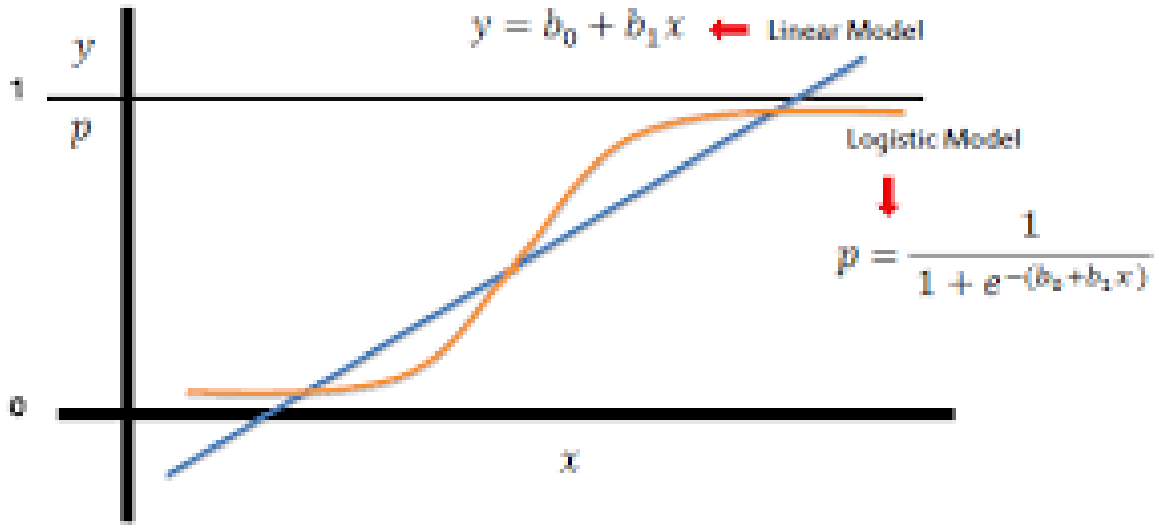
Both Linear and Logistic Regression can be used to model the *probability* of a discrete response (Y), such as *default* on a credit-card payment, as a function of a predictor variable (X), such as *balance* of credit card.

P(Y= true | X) re-written as P(X)

It is possible to represent this as a linear function

$$\mathbf{P(X) = b_0 + b_1X}$$

Problems include: (a) probability changing *too gradually* for a clear-cut decision; (b) probability returning values *below* 0 and *above* 1!



Linear Vs. Logistic Regression for Classification

The solution to both of these problem is Logistic Regression. This uses the function:

$$P(X) = e^{b_0 + b_1X} / (1 + e^{b_0 + b_1X}) \quad \text{equation (a)}$$

re-arranged, $P(X) / (1-P(X)) = e^{b_0 + b_1X}$

$\log_e [P(X) / (1-P(X))] = b_0 + b_1X$

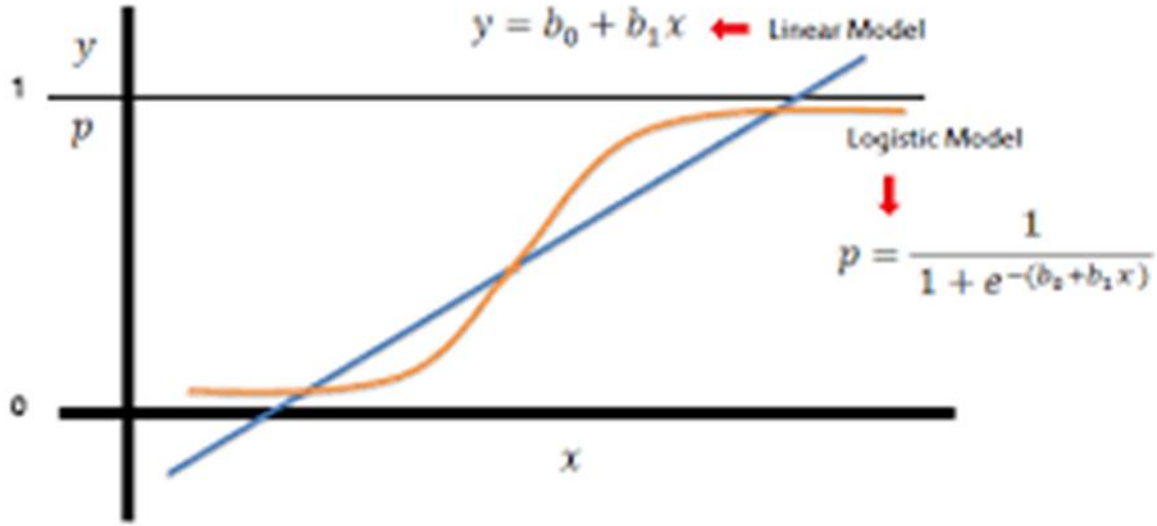
This 'logit' model is **linear** in X

Maximal likelihood is achieved by setting the b_0 and b_1 that would maximize the value returned by the formula:

$$\pi_{i,y_i=1} (P(x_i)) * \pi_{i,y_i=0} (1-P(x_i))$$

Where P(xi) is expressed as equation (a) above. The idea is we want to maximize the values of P(Xi) when Yi=1 *and* minimize them when Yi=0.

Various maximization algorithms can be used to find optimal values for b_0 and b_1 (not shown) but in any case, once these values are found, equation (a) is complete (and can be plotted as seen opposite).



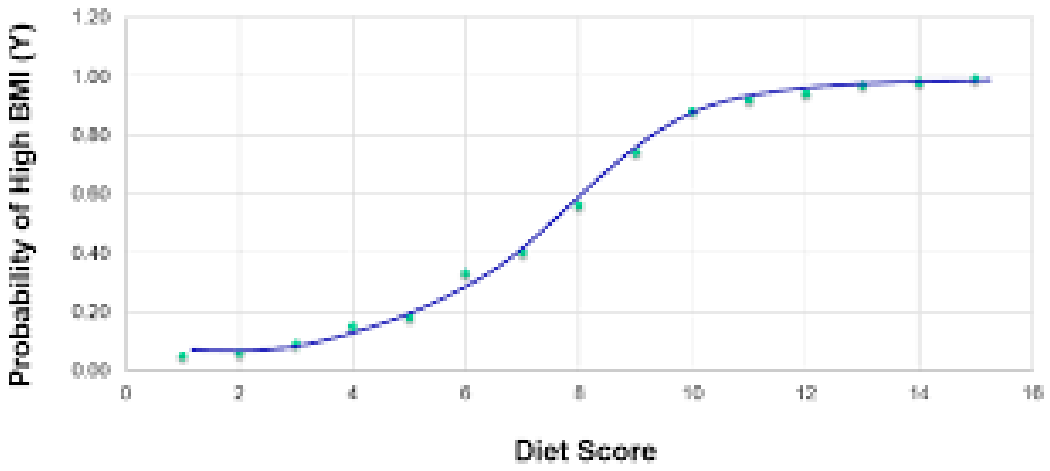
Making Predictions

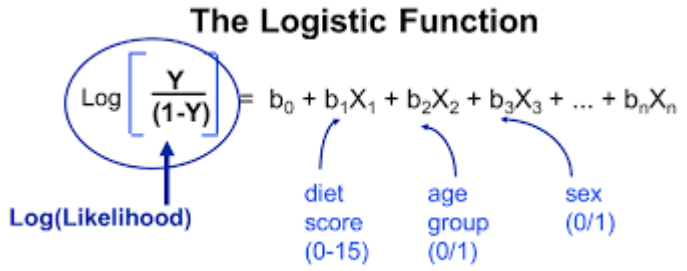
It is possible to make predictions using a single-predictor logistic model; just plug in a value for the predictor variable (say, X=Diet_Score=7) and it will return a prediction (e.g., Y=P(high BMI*)=0.40). However, it is even better to include a **set of predictors** (say n in total) **for a more accurate prediction**. In this case, formula (a) becomes:

$$P(X) = e^{b0 + b1X + b2X + .. + bnX +} / (1 + e^{b0 + b1X + b2X + .. + bnX})$$

Or equivalently, $\log_e [P(X) / (1-P(X))] = b0 + b1X + .. + bnX$ giving us real-world models such as ==========➔

Which cannot be plotted on a flat page!





Tree-based Classification

Trees are a way of describing a sequence of decisions that allow one to partition the predictor space into regions R_1, R_2, .., R_n

For every observation in R_j, let *prediction* ($\mathbf{y}_{Rj}$) = mean/median of all responses in R_j

Naturally, we would like to create predictions that minimize, RSS, where

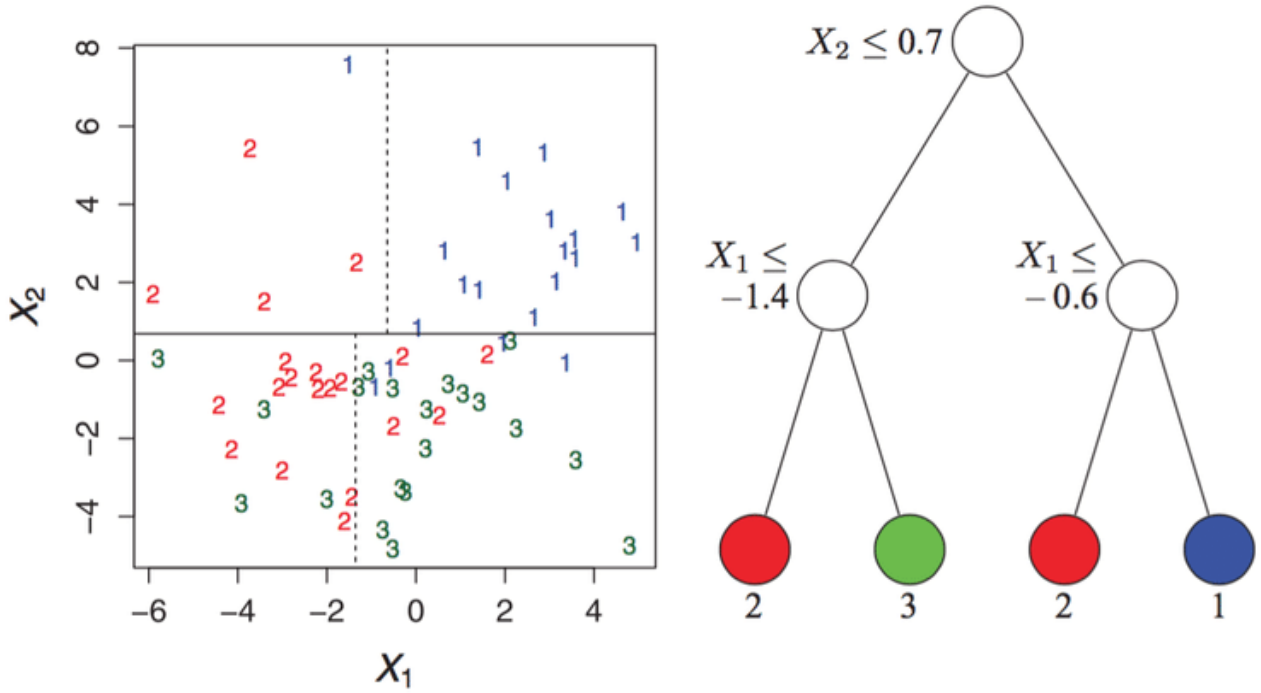
$$\text{RSS} = \sum_{j=1}^{J} \sum_{i \text{ in } Rj} (y_i - \mathbf{y}_{Rj})^2$$

Where y_i is the *actual* value of training response *i*.

In the example here, the classifier responds with one of three classes: 1 (blue), 2 (red) and 3 (green).

The predictor variables are X_1 and X_2.

Continued on following slide



Tree-based Classification Cont.

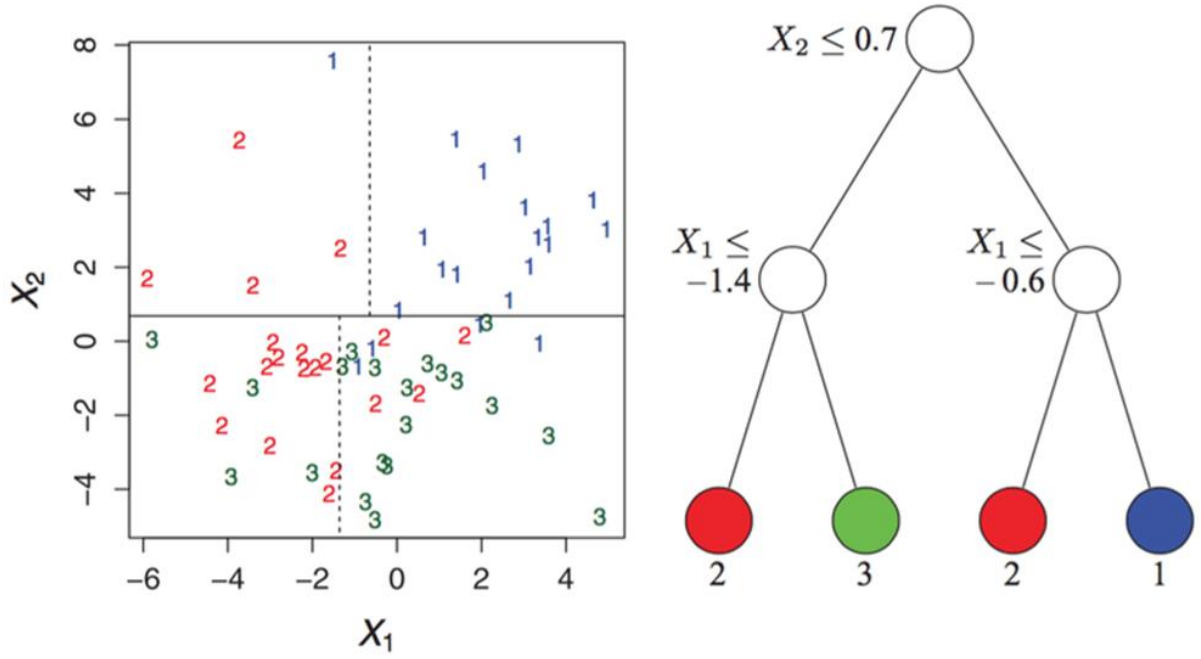
To form a classification tree, the full space (X_1 x X_2) - where X_1 is in [-6, 5] and X_2 is in [-5, 8] - is partitioned along one dimension (here, first along X_2) and hence, along X_1.

This is continued, recursively, until either

(1) All sub-spaces (= regions) contain instances of a *single class* or, more likely ..

(2) A more sophisticated termination criteria is satisfied (e.g., a *measure of purity*: later)

Once the tree is formed, training is complete, and the tree can be used to classify unseen instances, by starting at the root of the tree and going down the branch that satisfies the node condition, until a leaf node is reached: this returns a class prediction.



Measures of Node Purity (min disorder = max info gain, when splitting)

Let $\mathbf{p_{mk}}$ be the *proportion* (out of 1) of training instances in region m from class k. For example, in the region presented by the root node (i.e., whole space),

$\mathbf{p}_{blue}$ = 13/(13+22) = ~ 0.37

(1) **Gini Index** (G) =

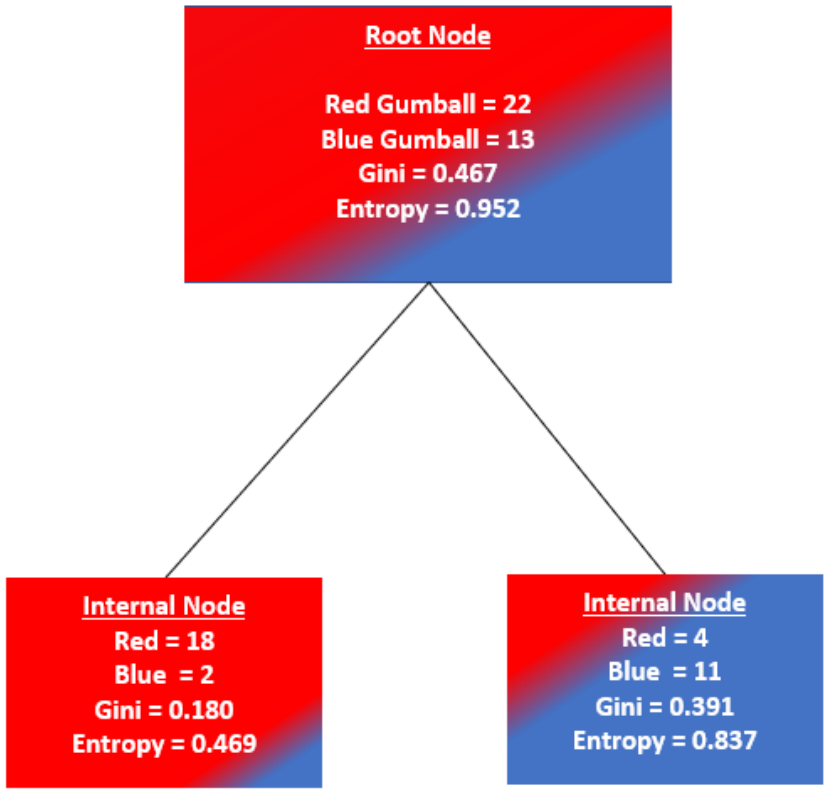
$$1 - \sum_{k=1}^{K} p_{mk}{}^2$$

(2) **Entropy** (D) =

$$- \sum_{k=1}^{K} p_{mk} \log_2(p_{mk})$$

The example numerically presents the values for both the Gini index and Entropy for all of the regions defined by their corresponding nodes;

You can visibly compare how many instances belong to one class (red) vs. the other (blue) by viewing the proportion of every node coloured with red vs. blue.



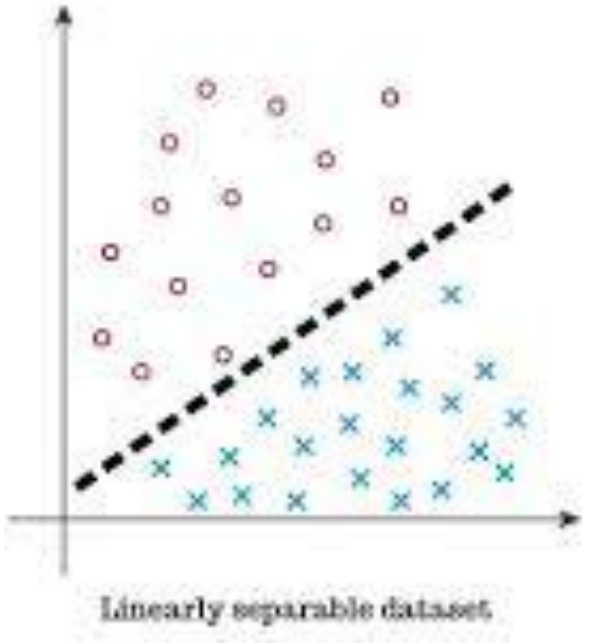
Classification Trees vs. Linear Models

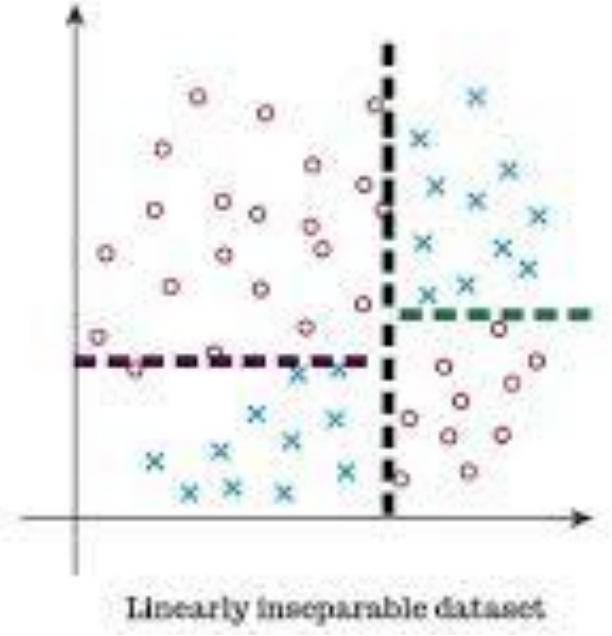
Advantages of Trees:

- Easy to explain
- Mirror human decision making
- Easily displayed graphically
- Capable of dealing with non-linearly separable instances

Disadvantages:

- Due to their orthogonal-to-axes space partitioning approach, they can return low accuracies or highly branched trees
- Not very robust, as small amounts of misclassified training instances can result in totally wrong partitions and hence, bad test accuracies





Possible Improvements: Bagging as an Example

Bagging Algorithm =

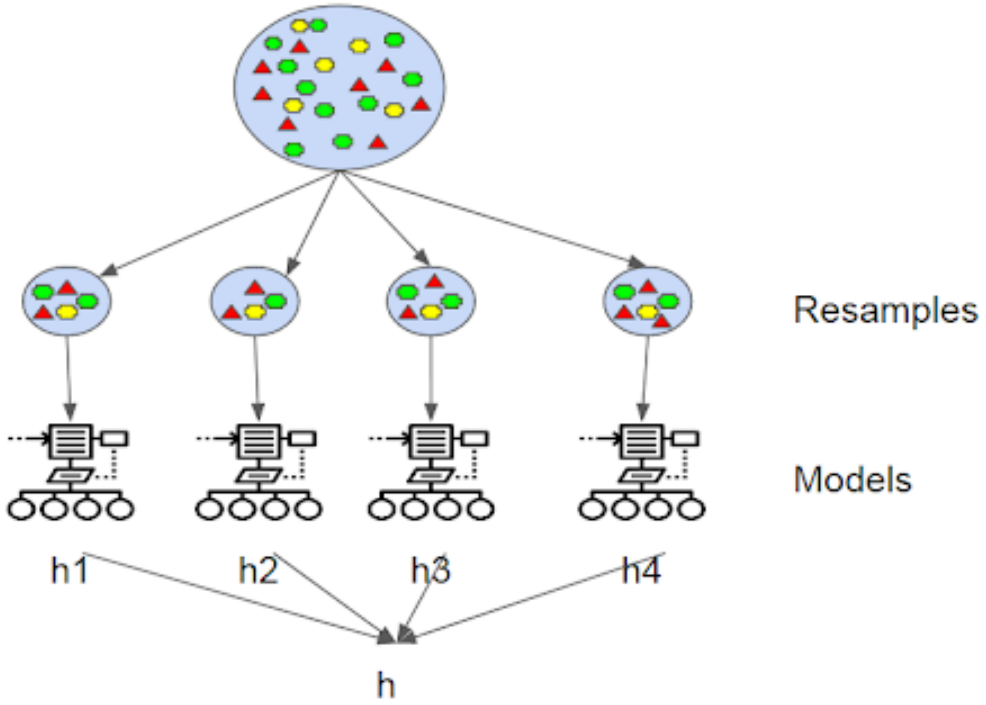
(1) Form a *new* sample of instances of size N by ***uniform re-sampling*** of the original sample of size N;

(2) Build a classification tree using this new sample;

(3) Repeat steps 1-2, M times, resulting in ***M* classification trees**

(4) When classifying, use *all M classification trees* to arrive at a *list* of M class predictions (per test instance);

(5) Assign the test instance the class prediction with the ***greatest frequency*** in the list of M predictions OR ***average the predictions*** (if they are continuous)



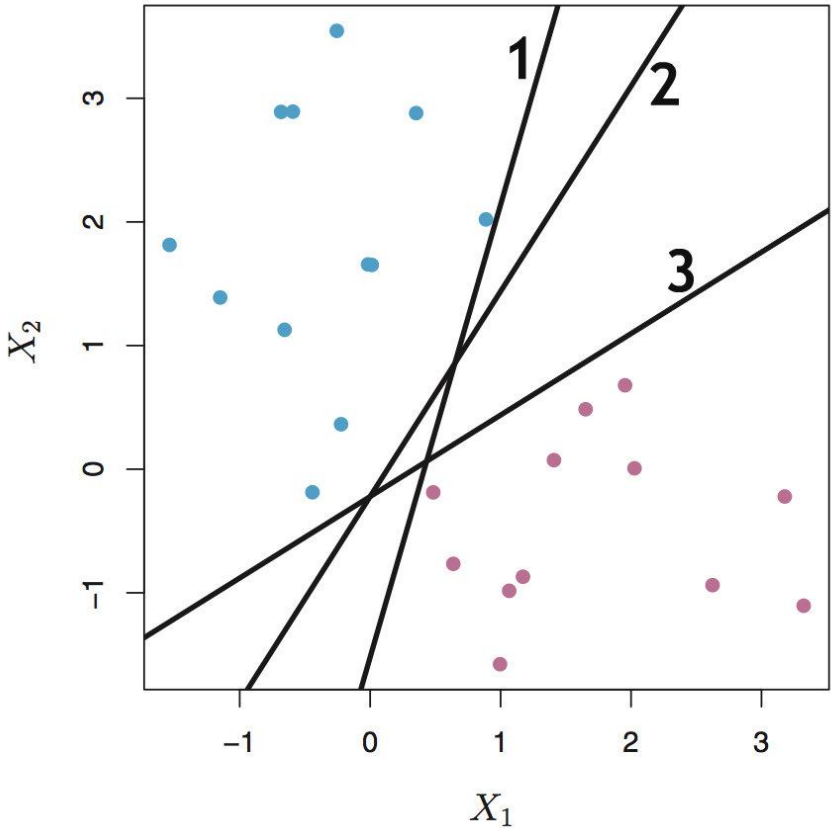
Maximum Margin Classifier

Maximal Margin Classifier requires classes to be separable by a linear boundary.

A hyperplane in n dimensions is a linear function in n variables (+ an offset) that divides the hyperspace into two halves:

$$f(\mathbf{X}) = b_0 + b_1 x_1 + b_2 x_2 + .. + b_n x_n = 0$$

In two dimensions, a hyperplane is just a line; all lines in the figure opposite divide the X_1 x X_2 space into two sub-spaces, where each of the sub-spaces contains instances of only one class (red or blue).



Maximum Margin Classifier Cont.

- Lines that fall on a line satisfy the equation, while pints, say **R** (a blue point) and **S** (a red one) fall either above or below the line, hence satisfying:

$$f(\mathbf{R}) = b_0 + b_1 r_1 + b_2 r_2 + .. + b_n r_n > 0$$

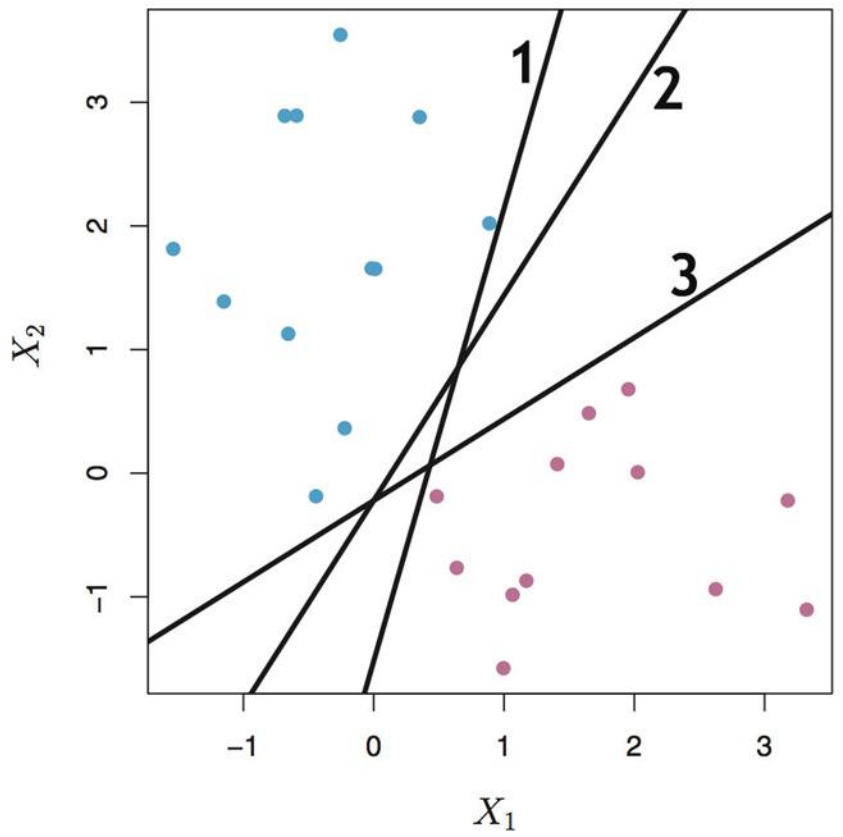
and

$$f(\mathbf{S}) = b_0 + b_1 s_1 + b_2 s_2 + .. + b_n s_n < 0$$

- As such, the process of training can be described as:

Given a set of training points with known classes (= colours);

Find a set of coefficient (b_0 .. b_n) that satisfy the first equation for *all* blue training points, and satisfy the bottom equation for *all* red training points.



Maximum Margin Classifier Cont.

From a mathematical perspective, if we call the response variable, Y, and set Y to 1 for red points, and -1 for blue points, then:

$$f(\mathbf{R}) = b_0 + b_1 r_1 + b_2 r_2 + .. + b_n r_n > 0 \quad \text{and } Y=+1$$

and

$$f(\mathbf{S}) = b_0 + b_1 s_1 + b_2 s_2 + .. + b_n s_n < 0 \quad \text{and } Y=-1$$

Both sides of the first equation multiplied by +1 results in no change in the equation, while if both sides of the bottom equation are multiplied by -1 then this will flip the inequality sign (from < to >) without affecting anything else. Hence, this mathematical manipulation results in two identical equations or - put another way - one equation of the form:

$$(b_0 + b_1 x_1 + b_2 x_2 + .. + b_n x_n) \; Y > 0$$

Where **X** stands for any point and Y is the numerical class label: +1 or -1.

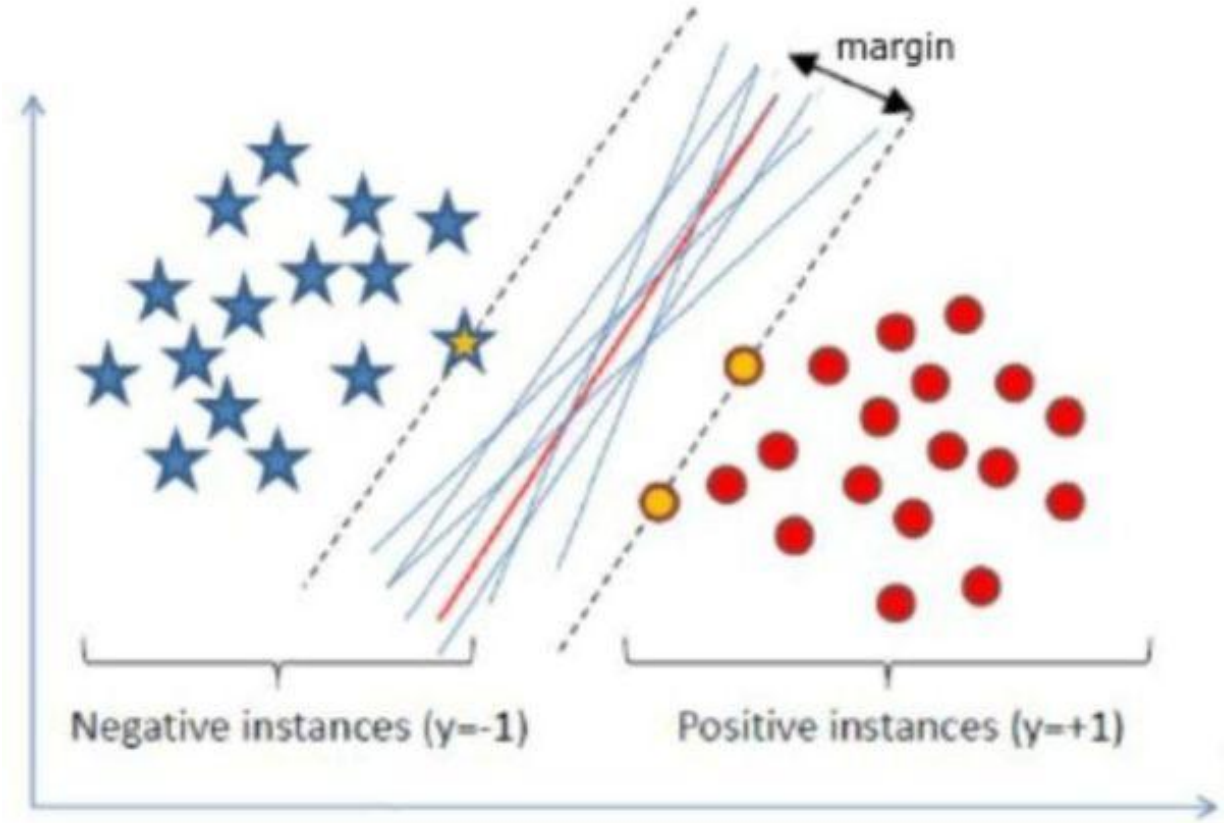
Maximal Margin

The Maximal Margin Classifier is defined by the hyperplane (here, line) that is *farthest* from all training instances.

All lines between the positive and negative instances are *legitimate* hyperplanes dividing the space into pure sub-spaces.

However, **only one of them (in red) has the *maximum* (sum of) distances from all the positive and negative training points.**

The *margin* is defined as the distance between this maximum separation line and the closest points to the line.



Maximal Margin Classifier

The MMC uses the same equation we've seen so far, but adding a margin (*M*), which we wish to maximize, and ensuring that all distances (from the separating hyperplane) are shortest (= orthogonal). Hence,

(1) Using an optimization algorithm, find those coefficients (b_i) that maximize *M* in equation

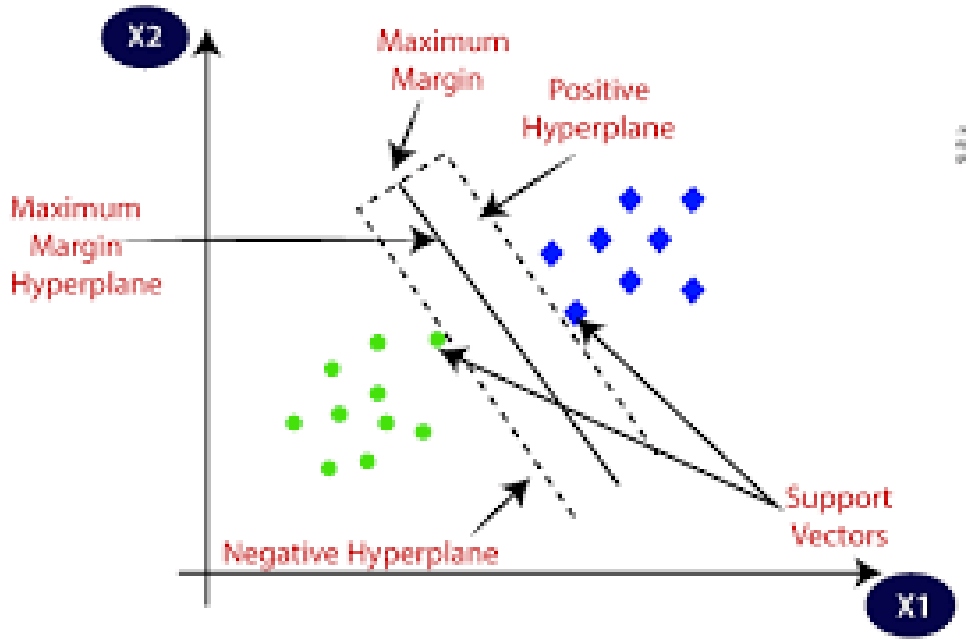
$$(b_0 + b_1 x_1 + b_2 x_2 + .. + b_n x_n) Y > 0 + M$$

Over all training instances (**X**,Y), and, **X** are the predictors (= coordinates in hyperspace), and Y is the assigned class label (+1 or -1);

(2) Ensure that the measured distances are shortest by asserting that:

$$\sum_{i=1}^{n} b_i^2 = 1$$

Note: the *support vectors* (or points) are defined as those points that have distances from the separation plane = the margin *M*.

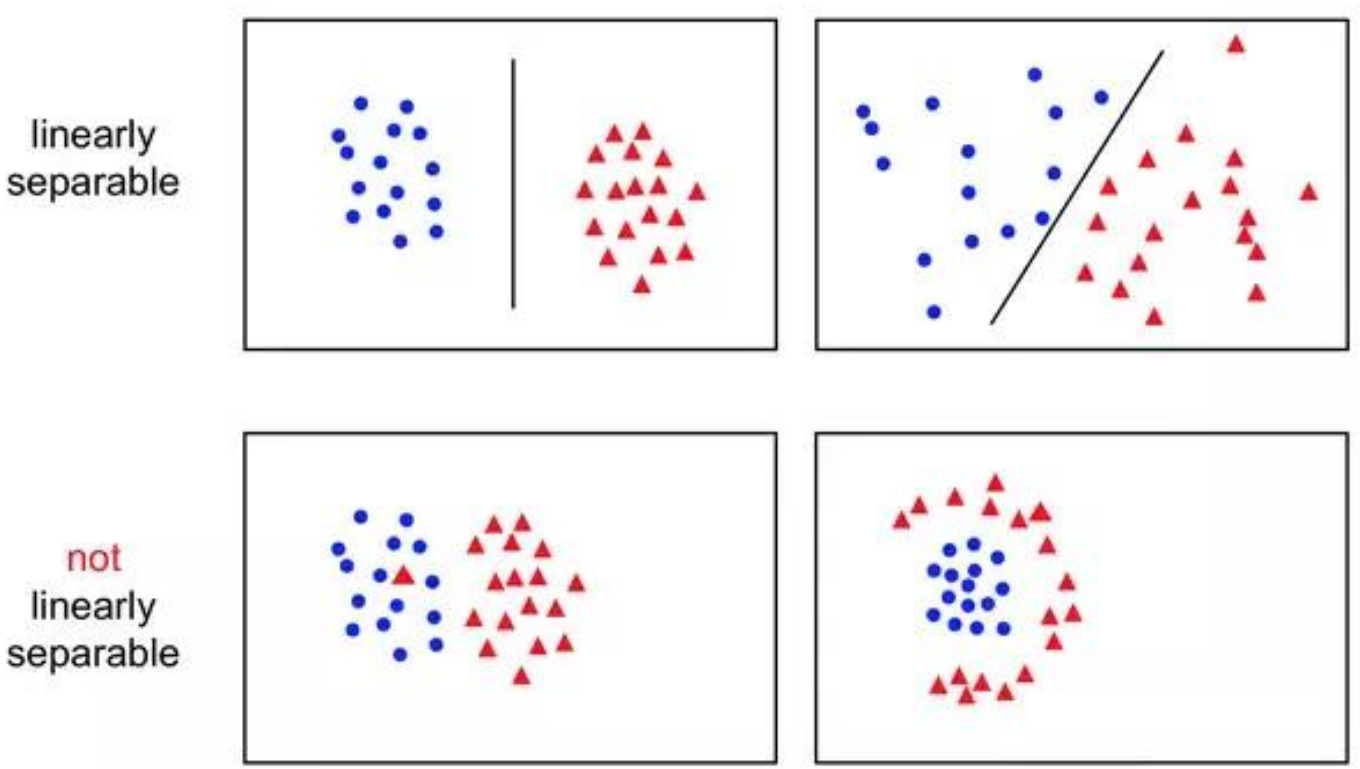


Support Vector Classifiers

In some cases, it is impossible to separate the space into two pure sub-spaces, because the points are arranged so.

In these cases, either a different kind of classifier should be used OR we must tolerate a certain amount of misclassification.

Support Vector Classifiers are such classifiers, as they have a tunable parameters C that control a certain amount of acceptable error.



Support Vector Classifiers Cont.

We now use the same equation above, but we add error (or slack) terms **e** allowing individual observations to be on the wrong side of the separation hyper-plane. As such, the news set of equations becomes:

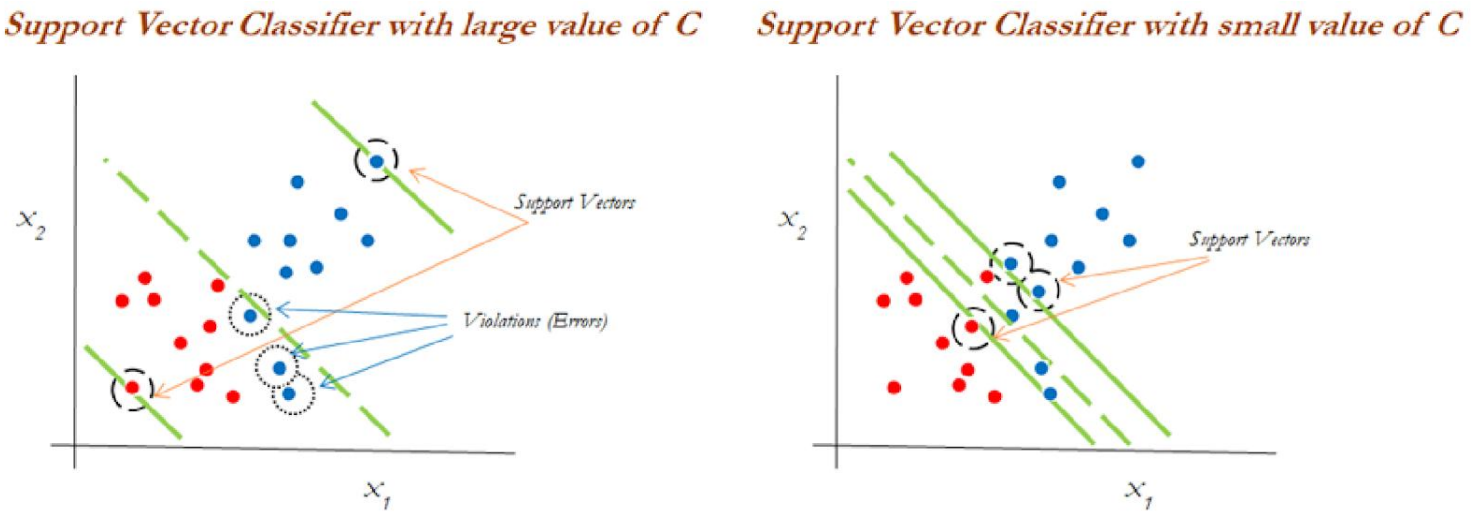
Choose coefficients b_i to *maximize M*

subject to $\sum_{i=1}^{n} b_i^2 = 1$ and

$(b_0 + b_1 x_1 + b_2 x_2 + .. + b_n x_n) \; Yi > M (1-e_i)$

where $e_i > 0$ and $\sum_{i=1}^{n} e_i <= C$

C is (usually) a positive tunable parameter.



Clustering vs. Classification

Classification is a process by which one assigns a class, usually a discrete label such as: 'sick', 'healthy' and 'suspect', to an instance, often represented by a point in a multi-dimensional space.

Classification usually goes through two phases:

Phase 1: a machine is trained (learns) from labeled instances how to divide the space of all possible instances (seen and unseen);

Phase 2: the machine is then applied to unseen instances that are unlabeled in order to decide which part of the pace it belongs to, i.e., what label it should have

Clustering is a process by which one places unlabeled instances (points) into mutually exclusive groups (or clusters) that have certain attributes in common, such as gender (male or female) or height (short, tall) or intelligence (below average, above average).

Clustering can be placed into several broad categories such as: centroid-based clustering (e.g., k-Means), connectivity-based clustering, density-based clustering, distribution-based clustering and even fuzzy clustering.

The common property of all clustering methods is that the points are *not* labeled; the cluster they're placed in is a function of their own attributes, not an attached label imposed on them by some 'intelligent' observer.

Clustering Example: k-Means

Step 1: *Given K* clusters of points (say 3), each point is *randomly assigned* to exactly of the 3 clusters;

Step 2: *For each* cluster, compute the *centroid* (mean) of all the points that belong to it;

Step 3: *Calculate* the *distances* between each and every point and the *3* centroids of the various clusters;

Step 4: *Assign* each point to the cluster, whose centroid is closest to it: in some cases this will result in the *re-assignment* of a point to a different cluster than the one it was in;

Step 5: IF in step 4, none of the points were re-assigned THEN terminate, ELSE *iterate* by going to Step 2.