

Classification-Prediction: Back Propagation & Regression



Dr. Manish Kumar

Associate Professor

Chair: Data Analytics Lab & M.Tech (Data Engg.)

Department of Information Technology

Indian Institute of Information Technology-Allahabad, Prayagraj



Back Propagation



Basics of Neural Network

- What is a Neural Network
- Neural Network Classifier
- Data Normalization
- Neuron and bias of a neuron
- Single Layer Feed Forward
- Limitation
- Multi Layer Feed Forward
- Back propagation

Neural Networks

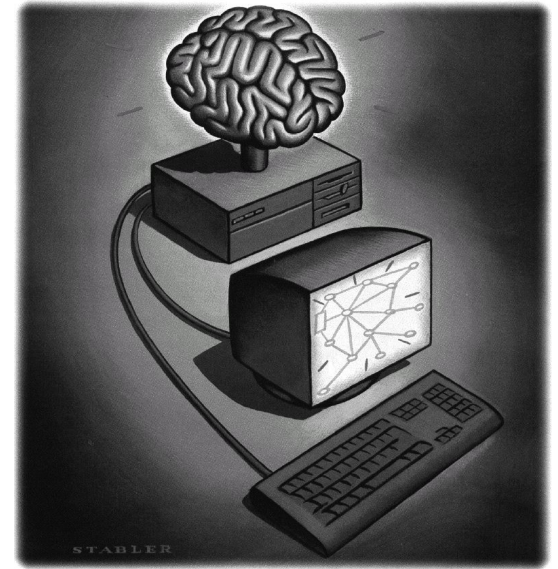
What is a Neural Network?

- Biologically motivated approach to machine learning

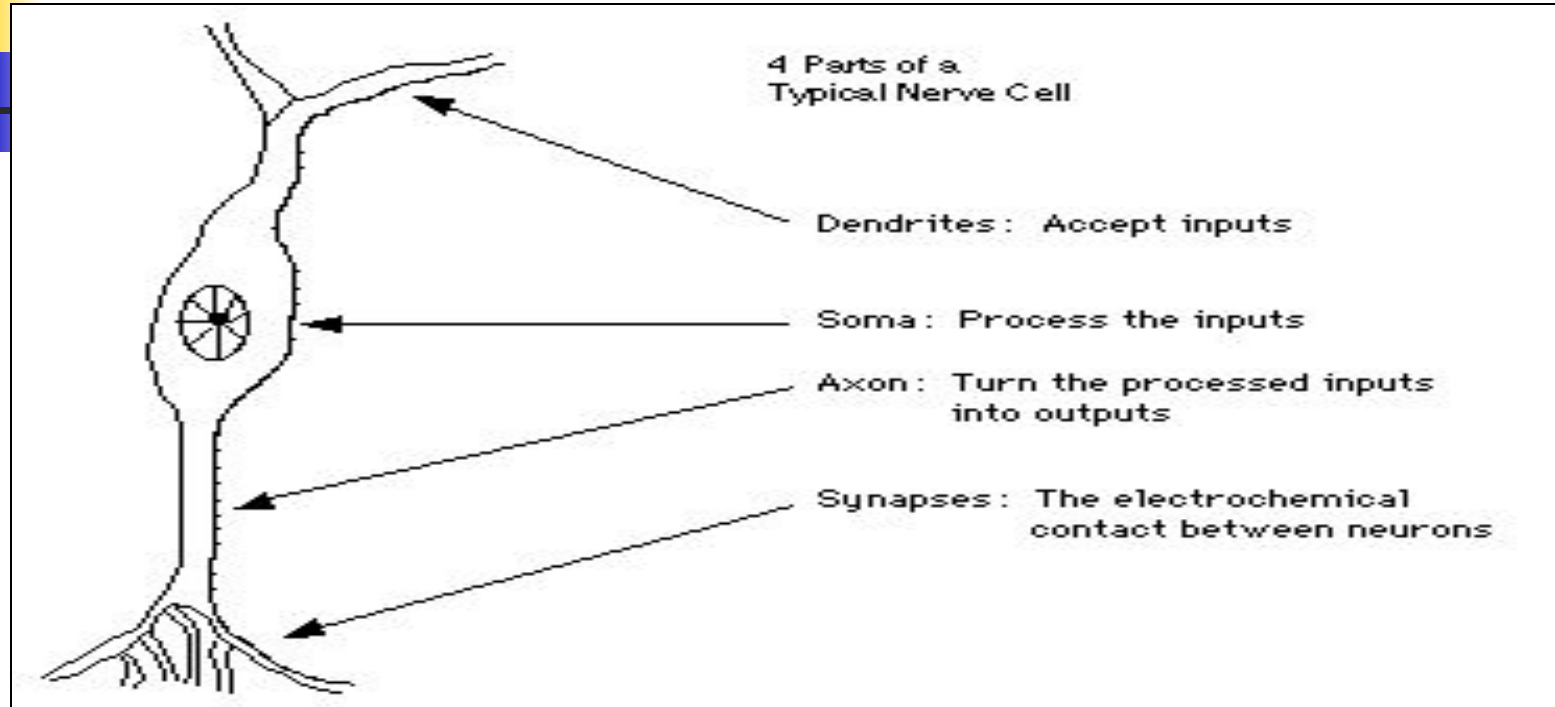
Similarity with biological network

Fundamental processing elements of a neural network is a neuron

1. Receives inputs from other source
2. Combines them in someway
3. Performs a generally nonlinear operation on the result
4. Outputs the final result

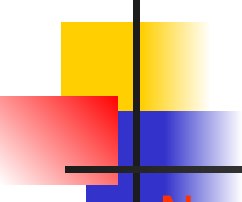


Similarity with Biological Network



- Fundamental processing element of a neural network is a neuron
- A human brain has 100 billion neurons
- An ant brain has 250,000 neurons

Neural Network

- 
-
- **Neural Network** is a set of connected INPUT/OUTPUT UNITS, where each connection has a WEIGHT associated with it.
 - **Neural Network** learning is also called CONNECTIONIST learning due to the connections between units.
 - It is a case of SUPERVISED, INDUCTIVE or CLASSIFICATION learning.



Neural Network

- **Neural Network** learns by adjusting the weights so as to be able to correctly classify the training data and hence, after testing phase, to classify unknown data.
- **Neural Network** needs long time for training.
- **Neural Network** has a high tolerance to noisy and incomplete data



Neural Network Classifier

- **Input: Classification data**
It contains classification attribute
- Data is divided, as in any classification problem.
[Training data and Testing data]
- **All data must be normalized.**
(i.e. all values of attributes in the database are changed to contain values in the interval $[0,1]$ or $[-1,1]$)
Neural Network can work with data in the range of $(0,1)$ or $(-1,1)$
- **Two basic normalization techniques**
 - [1] Max-Min normalization
 - [2] Decimal Scaling normalization



Data Normalization

[1] Max- Min normalization formula is as follows:

$$v' = \frac{v - \min A}{\max A - \min A} (\text{new_max } A - \text{new_min } A) + \text{new_min } A$$

[minA, maxA , the minimum and maximum values of the attribute A

max-min normalization maps a value v of A to v' in the range $\{\text{new_minA}, \text{new_maxA}\}$]

Example of Max-Min Normalization

Max- Min normalization formula

$$v' = \frac{v - \min A}{\max A - \min A} (\text{new_max } A - \text{new_min } A) + \text{new_min } A$$

Example: We want to normalize data to range of the interval [0,1].

We put: **new_max A= 1, new_minA =0.**

Say, max A was 100 and min A was 20 (That means maximum and minimum values for the attribute).

Now, if $v = 40$ (If for this particular pattern , attribute value is 40), v' will be calculated as , $v' = (40-20)$

$\times (1-0) / (100-20) + 0$

$$\Rightarrow v' = 20 \times 1/80$$

$$\Rightarrow v' = 0.4$$

Decimal Scaling Normalization



[2]Decimal Scaling Normalization

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A.

$$v' = \frac{v}{10^j}$$

Here j is the smallest integer such that $\max|v'| < 1$.

Example :

A – values range from -986 to 917. Max $|v|$ = 986.

$v = -986$ normalize to $v' = -986/1000 = -0.986$

One Neuron as a Network

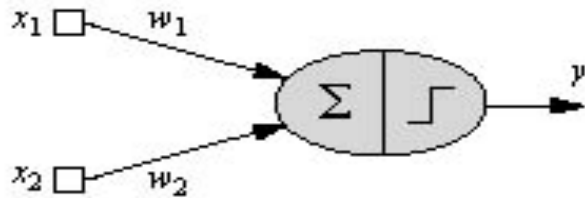


Fig1: an artificial neuron

- Here x_1 and x_2 are normalized attribute value of data.
- y is the output of the neuron , i.e the class label.
- x_1 and x_2 values multiplied by weight values w_1 and w_2 are input to the neuron x .
- Value of x_1 is multiplied by a weight w_1 and values of x_2 is multiplied by a weight w_2 .
- Given that
 - $w_1 = 0.5$ and $w_2 = 0.5$
 - Say value of x_1 is 0.3 and value of x_2 is 0.8,
 - So, weighted sum is :
 - $\text{sum} = w_1 \times x_1 + w_2 \times x_2 = 0.5 \times 0.3 + 0.5 \times 0.8 = 0.55$



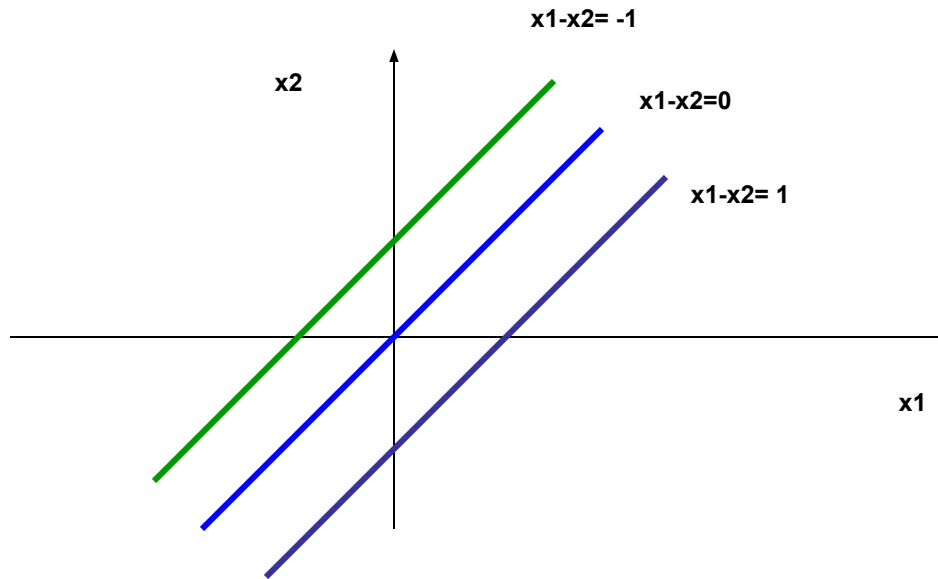
One Neuron as a Network

- The neuron receives the weighted sum as input and calculates the output as a function of input as follows :
- $y = f(x)$, where $f(x)$ is defined as
- $f(x) = 0$ { when $x < 0.5$ }
- $f(x) = 1$ { when $x \geq 0.5$ }
- For our example, x (weighted sum) is 0.55, so $y = 1$,
- That means corresponding input attribute values are classified in class 1.
- If for another input values , $x = 0.45$, then $f(x) = 0$,
- so we could conclude that input values are classified to class 0.

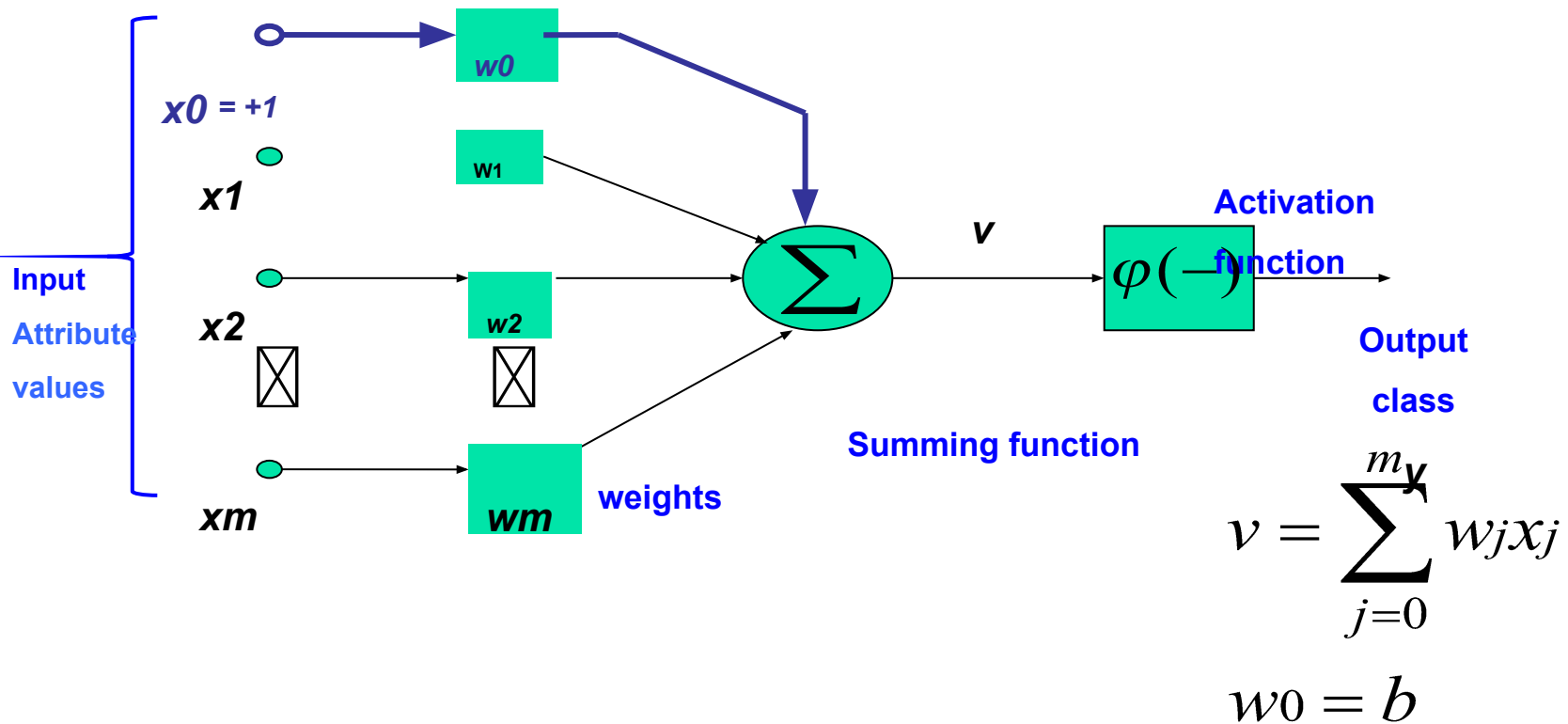
Bias of a Neuron

- We need the bias value to be added to the weighted sum $\sum w_i x_i$ so that we can transform it from the origin.

$$v = \sum w_i x_i + b, \text{ here } b \text{ is the bias}$$



Bias as extra input



Neuron with Activation



The neuron is the basic information processing unit of a NN. It consists of:

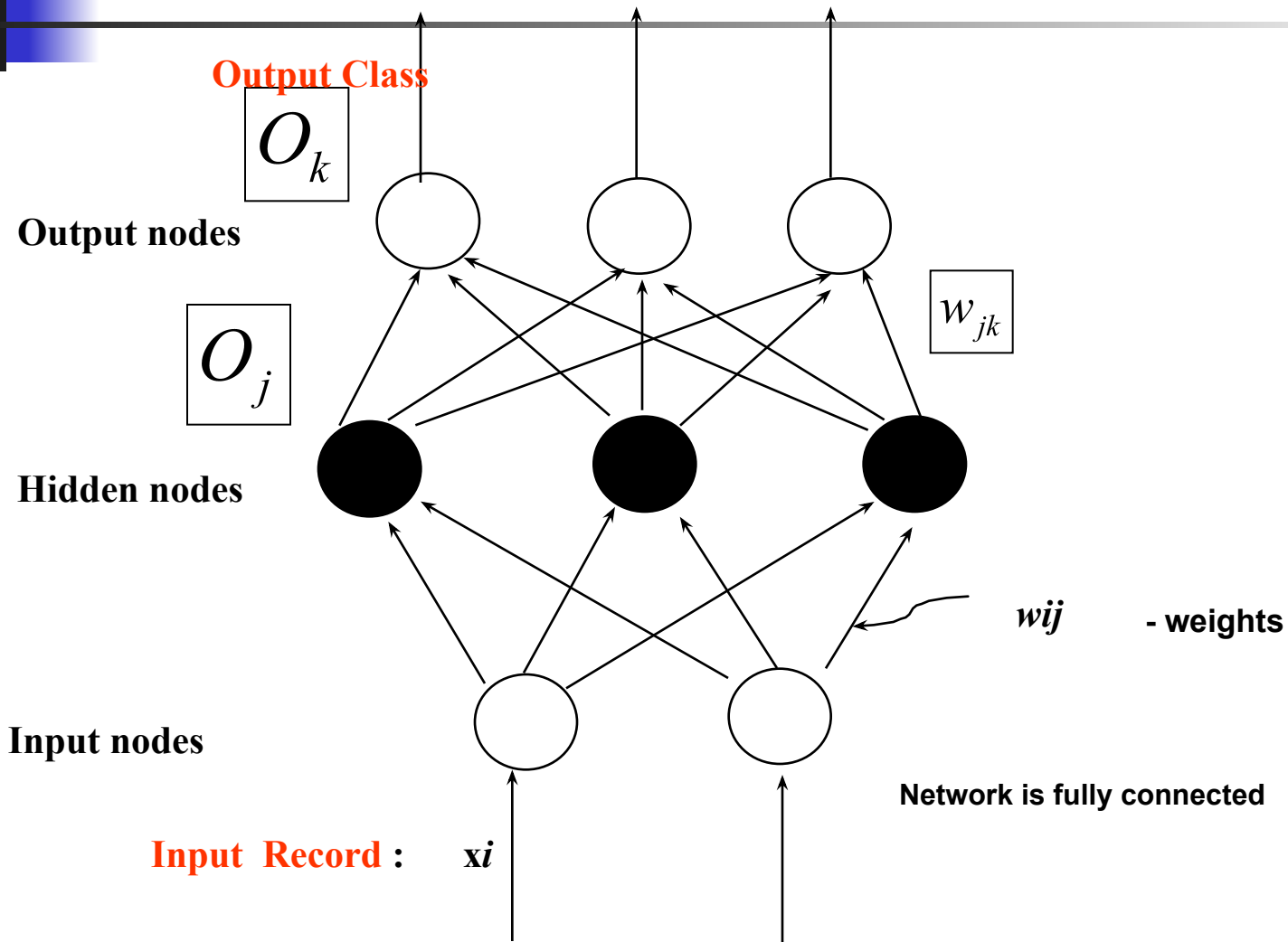
- 1 A set of **links**, describing the neuron inputs, with weights W_1, W_2, \dots, W_m
2. An **adder** function (linear combiner) for computing the weighted sum of the inputs (real numbers):

$$u = \sum_{j=1}^m w_j x_j$$

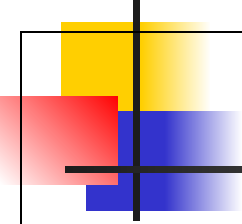
- 3 **Activation function** : for limiting the amplitude of the neuron output.

$$y = \varphi(u + b)$$

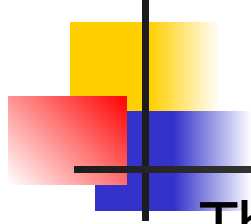
A Multilayer Feed-Forward Neural Network



Neural Network Learning

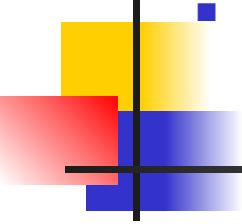
- 
- The inputs are fed simultaneously into the input layer.
 - The weighted outputs of these units are fed into hidden layer.
 - The weighted outputs of the last hidden layer are inputs to units making up the output layer.

Network



- The units in the hidden layers and output layer are sometimes referred to as **neurodes**, due to their symbolic biological basis, or as **output units**.
- A network containing two hidden layers is called a **three-layer** neural network, and so on.
- The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer.

A Multilayered Feed – Forward Network

- 
- **INPUT:** records without class attribute with normalized attributes values.
 - **INPUT VECTOR:** $X = \{x_1, x_2, \dots, x_n\}$
where n is the number of (non class) attributes.
 - **INPUT LAYER** – there are as many nodes as non-class attributes i.e. as the length of the input vector.
 - **HIDDEN LAYER** – the number of nodes in the hidden layer and the number of hidden layers depends on implementation.

A Multilayered Feed-Forward Network

- **OUTPUT LAYER** – corresponds to the class attribute.
- There are as many nodes as classes (values of the class attribute).

$$O_k$$

$k = 1, 2, \dots \text{\#classes}$

- Network is **fully connected**, i.e. each unit provides input to each unit in the next forward layer.

Classification by Back propagation



- *Back Propagation learns by iteratively processing a set of training data (samples).*
- For each sample, weights are modified to minimize the error between network's classification and actual classification.



Steps in Back propagation Algorithm

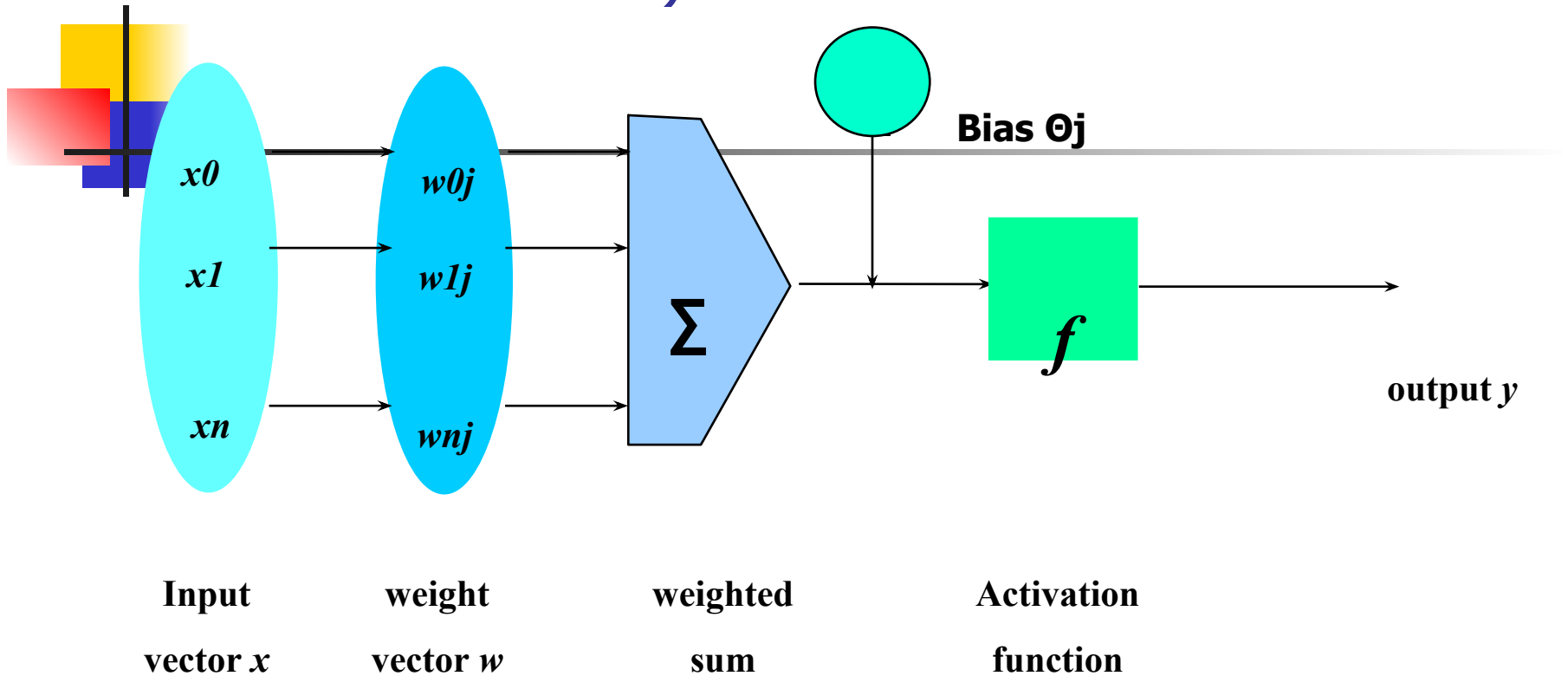
- STEP ONE: initialize the weights and biases.
 - The weights in the network are initialized to random numbers from the interval $[-1,1]$.
 - Each unit has a BIAS associated with it
 - The biases are similarly initialized to random numbers from the interval $[-1,1]$.
- STEP TWO: feed the training sample.



Steps in Back propagation Algorithm (cont..)

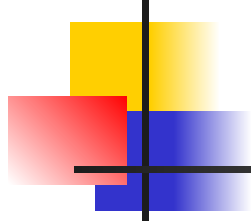
- **STEP THREE:** Propagate the inputs forward; we compute the net input and output of each unit in the hidden and output layers.
- **STEP FOUR:** back propagate the error.
- **STEP FIVE:** update weights and biases to reflect the propagated errors.
- **STEP SIX:** terminating conditions.

Propagation through Hidden Layer (One Node)



- The inputs to unit j are outputs from the previous layer. These are multiplied by their corresponding weights in order to form a weighted sum, which is added to the bias associated with unit j .
- A nonlinear activation function f is applied to the net input.

Propagate the inputs forward



■ For unit j in the input layer, its output is equal to its input, that is,

$$O_j = I_j$$

for input unit j .

● The net input to each unit in the hidden and output layers is computed as follows.

● Given a unit j in a hidden or output layer, the net input is

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

where w_{ij} is the weight of the connection from unit i in the previous layer to unit j ; O_i is the output of unit i from the previous layer;

$$\theta_j$$

is the bias of the unit



Propagate the inputs forward

- Each unit in the hidden and output layers takes its net input and then applies an **activation function**. The function symbolizes the activation of the neuron represented by the unit. It is also called a **logistic, sigmoid, or squashing function**.
- Given a net input I_j to unit j , then

$$O_j = f(I_j),$$

the output of unit j , is computed as $O_j = \frac{1}{1 + e^{-I_j}}$

Back propagate the error



When reaching the Output layer, the error is computed and propagated backwards.

For a unit k in the output layer the error is computed by a formula:

$$Err_k = O_k (1 - O_k) (T_k - O_k)$$


Where O_k – actual output of unit k (computed by activation function.

$$O_k = \frac{1}{1 + e^{-I_k}}$$

T_k – True output based of known class label; classification of training sample

$O_k(1-O_k)$ – is a Derivative (rate of change) of activation function.

Back propagate the error




The error is propagated backwards by updating weights and biases to reflect the error of the network classification .

- For a unit j in the hidden layer the error is computed by a formula:

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$$

where w_{jk} is the weight of the connection from unit j to unit k in the next higher layer, and Err_k is the error of unit k .

Update weights and biases



Weights are updated by the following equations, where l is a constant between 0.0 and 1.0 reflecting **the learning rate**, this learning rate is **fixed for implementation**.

$$\Delta w_{ij} = (l)Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

- **Biases** are updated by the following equations

$$\Delta \theta_j = (l)Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

Update weights and biases



■ We are updating weights and biases after the presentation of each sample.

■ This is called case updating.

- **Epoch** --- One iteration through the training set is called an epoch.
- **Epoch updating -----**
- **Alternatively, the weight and bias increments could be accumulated in variables and the weights and biases updated after all of the samples of the training set have been presented.**
- **Case updating is more accurate**

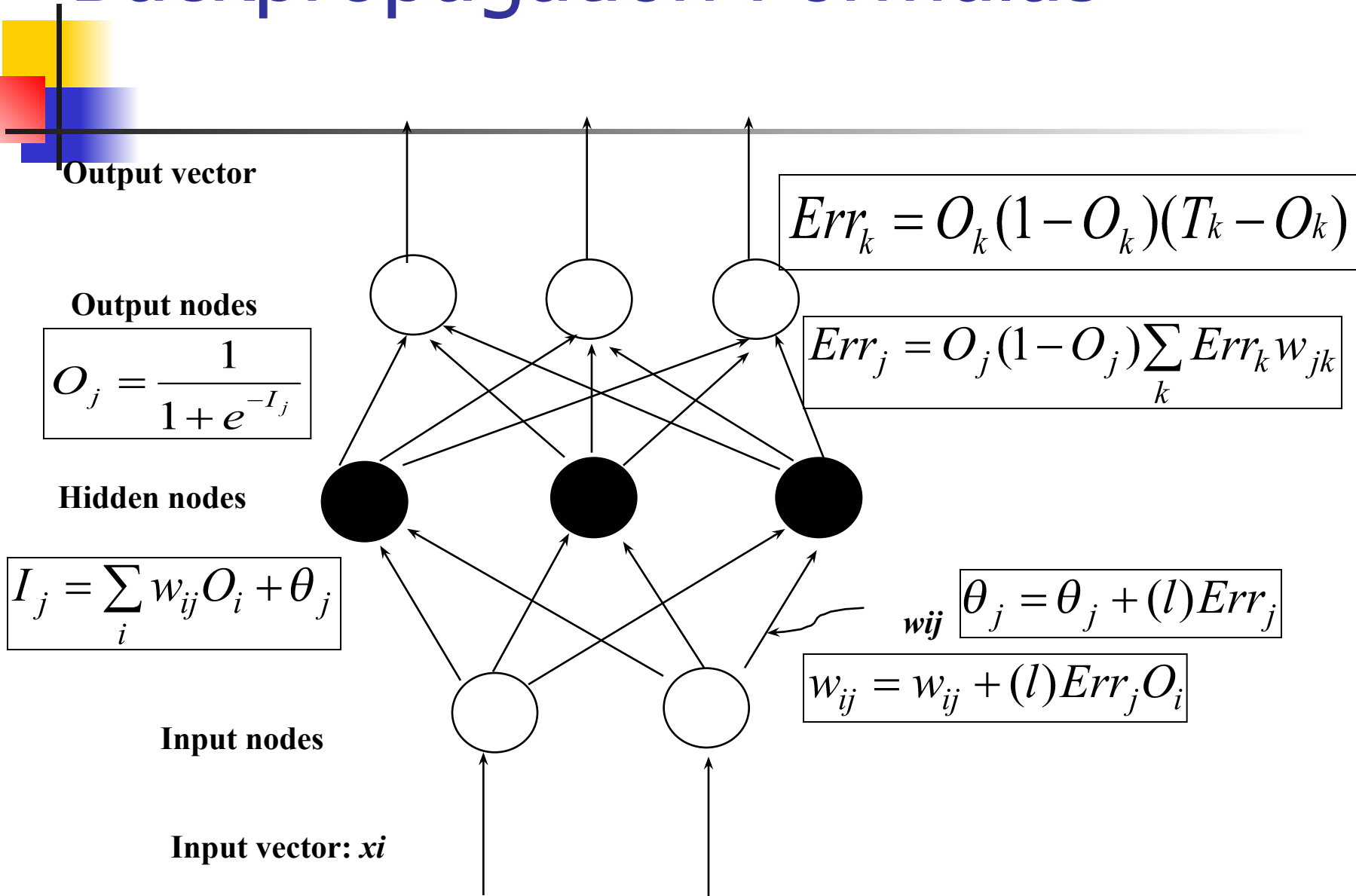


Terminating Conditions

- **Training stops**

- All Δw_{ij} in the previous epoch are below some threshold, or
- The percentage of samples misclassified in the previous epoch is below some threshold, or
- a pre specified number of epochs has expired.
- In practice, **several hundreds of thousands of epochs** may be required before the weights will converge.

Backpropagation Formulas



Example of Back propagation

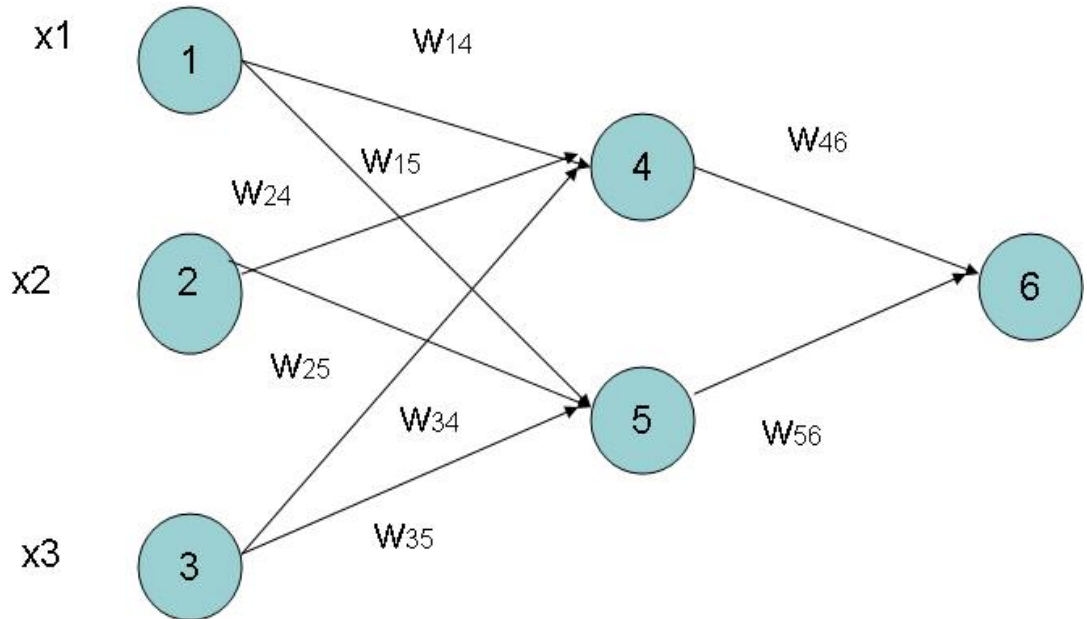
Input = 3, Hidden Neuron = 2

Output = 1

Initialize weights :

Random Numbers from -1.0 to 1.0

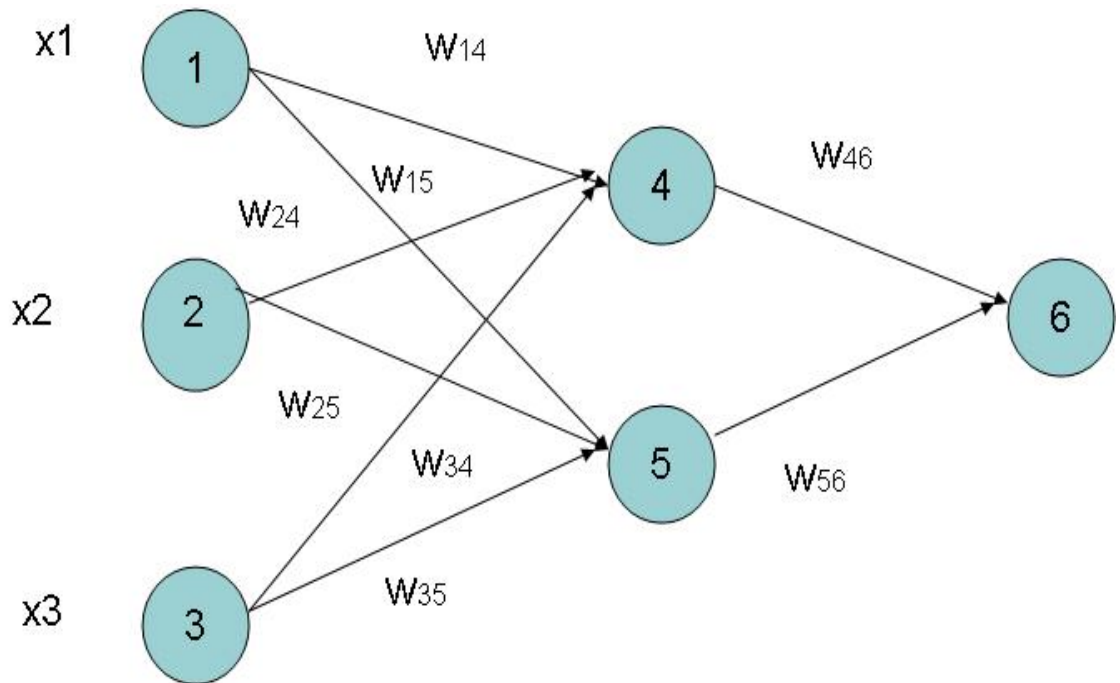
Initial Input and weight



x1	x2	x3	W_{14}	W_{15}	W_{24}	W_{25}	W_{34}	W_{35}	W_{46}	W_{56}
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2

Example (cont..)

- Bias added to Hidden
- + Output nodes
- Initialize Bias
- Random Values from -1.0 to 1.0
- Bias (Random)



θ_4	θ_5	θ_6
-0.4	0.2	0.1

Net Input and Output Calculation

Unit j	Net Input I_j	Output O_j
4	$0.2 + 0 + 0.5 - 0.4 = -0.7$	$O_j = \frac{1}{1 + e^{0.7}} = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$O_j = \frac{1}{1 + e^{-0.1}} = 0.525$
6	$(-0.3)0.332 - (0.2)(0.525) + 0.1 = -0.105$	$O_j = \frac{1}{1 + e^{0.105}} = 0.475$

Calculation of Error at Each Node

Unit j	Error j
6	$0.475(1-0.475)(1-0.475) = 0.1311$ We assume $T_6 = 1$
5	$0.525 \times (1 - 0.525) \times 0.1311 \times (-0.2) = 0.0065$
4	$0.332 \times (1 - 0.332) \times 0.1311 \times (-0.3) = -0.0087$

Calculation of weights and Bias Updating

Learning Rate $\eta = 0.9$

Weight	New Values
w₄₆	$-0.3 + 0.9(0.1311)(0.332) = -0.261$
w₅₆	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w₁₄	$0.2 + 0.9(-0.0087)(1) = 0.192$
w₁₅	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
.....similarlysimilarly
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
.....similarlysimilarly



Network Pruning and Rule Extraction

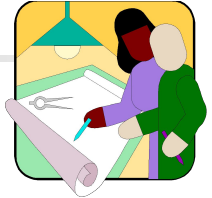
- Network pruning
 - Fully connected network will be hard to articulate
 - N input nodes, h hidden nodes and m output nodes lead to $h(m+N)$ weights
 - Pruning: Remove some of the links without affecting classification accuracy of the network



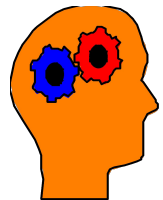
Applications

- Handwritten Digit Recognition
- Face recognition
- Time series prediction
- Process identification
- Process control
- Optical character recognition

Application-II



- Forecasting/Market Prediction: finance and banking
- Manufacturing: quality control, fault diagnosis
- Medicine: analysis of electrocardiogram data, RNA & DNA sequencing, drug development without animal testing
- Control: process, robotics





Regression



Recall: Covariance

$$\text{cov}(x, y) = \frac{\sum_{i=1}^n (x_i - \bar{X})(y_i - \bar{Y})}{n - 1}$$



Interpreting Covariance

$\text{cov}(X, Y) > 0 \rightarrow$ X and Y are positively correlated

$\text{cov}(X, Y) < 0 \rightarrow$ X and Y are inversely correlated

$\text{cov}(X, Y) = 0 \rightarrow$ X and Y are independent



Correlation coefficient

- Pearson's Correlation Coefficient is standardized covariance (unitless):

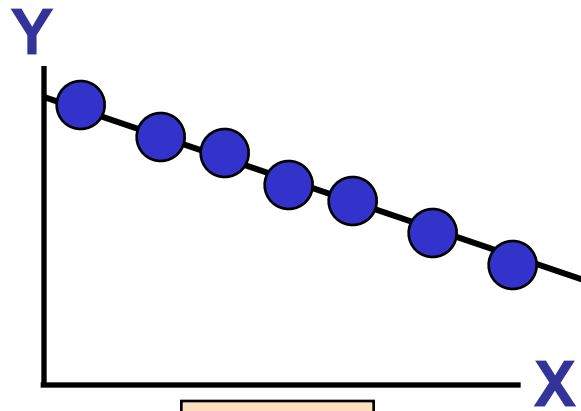
$$r = \frac{\text{covariance}(x, y)}{\sqrt{\text{var } x} \sqrt{\text{var } y}}$$



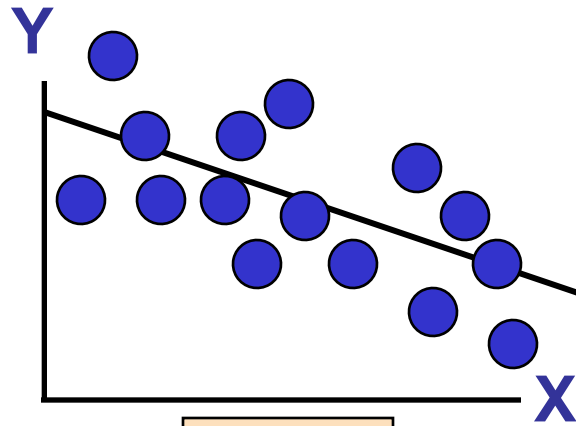
Correlation

- Measures the relative strength of the *linear* relationship between two variables
- Unit-less
- Ranges between -1 and 1
- The closer to -1 , the stronger the negative linear relationship
- The closer to 1 , the stronger the positive linear relationship
- The closer to 0 , the weaker any positive linear relationship

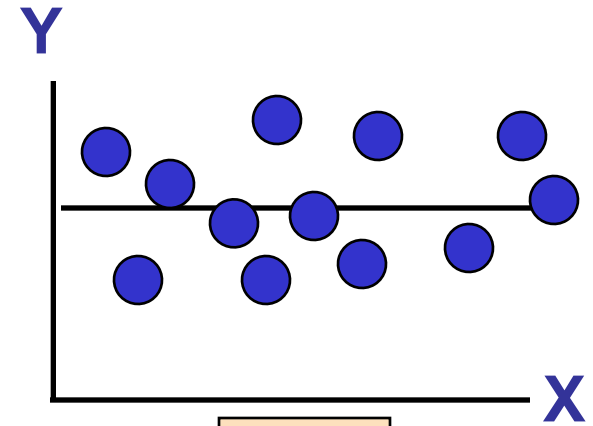
Scatter Plots of Data with Various Correlation Coefficients



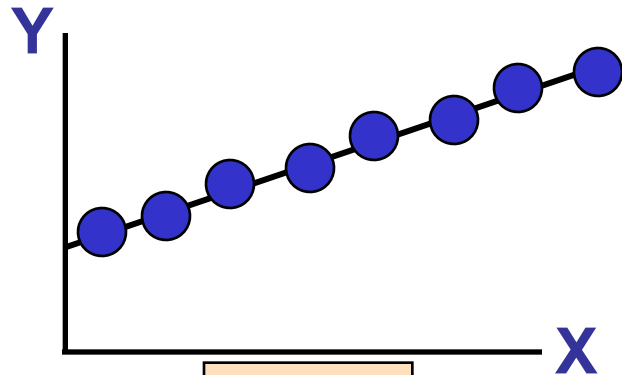
$$r = -1$$



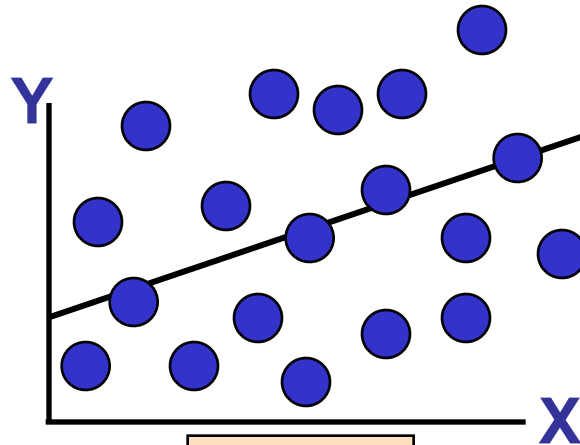
$$r = -.6$$



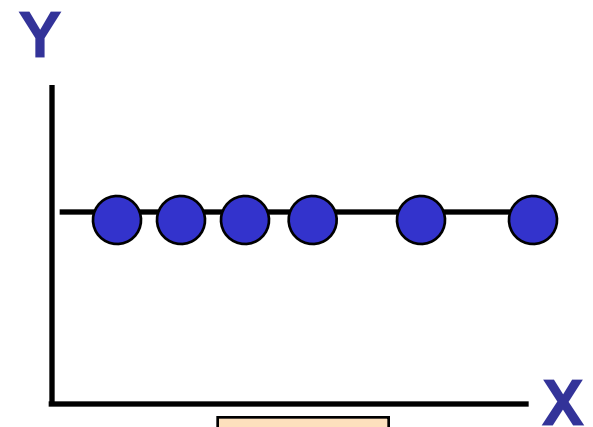
$$r = 0$$



$$r = +1$$



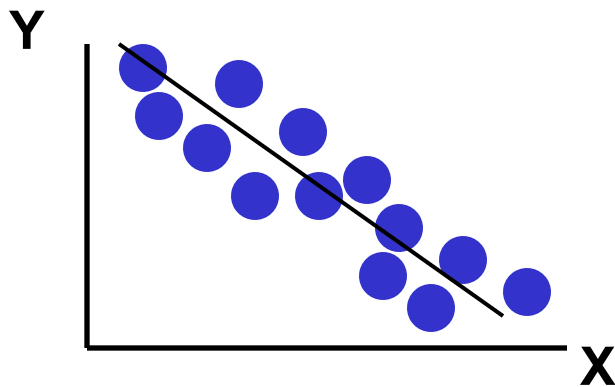
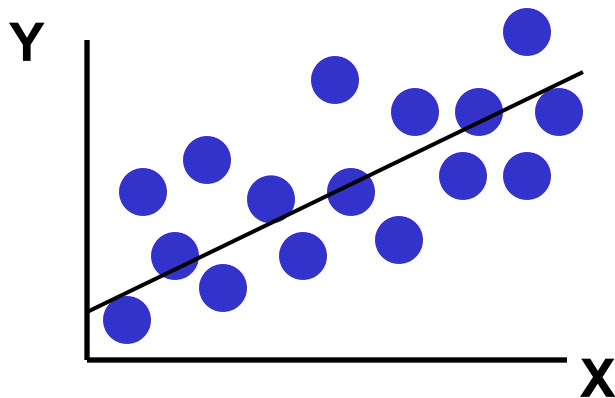
$$r = +.3$$



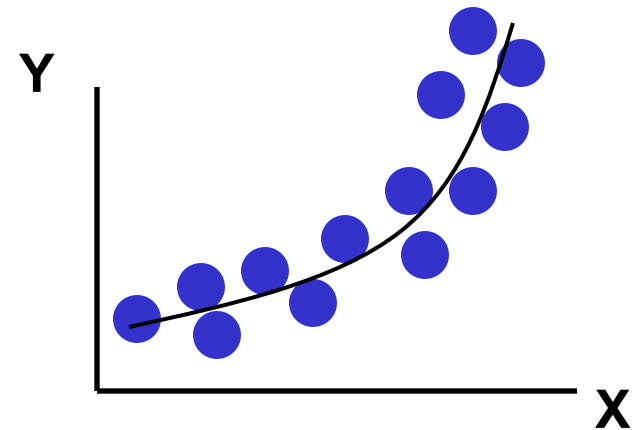
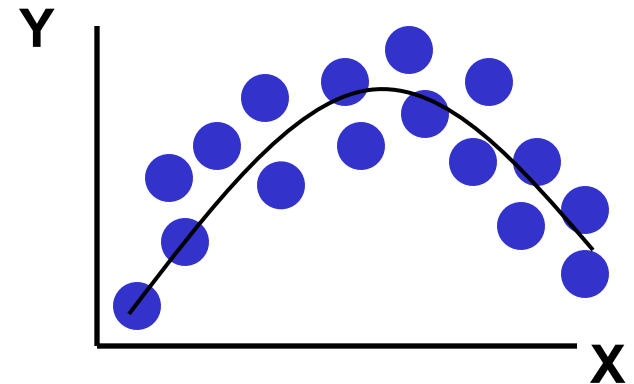
$$r = 0$$

Linear Correlation

Linear relationships

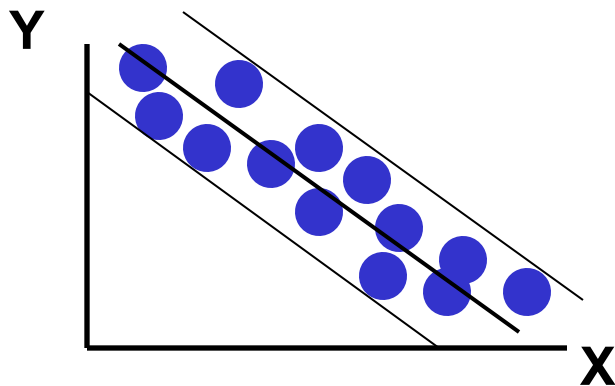
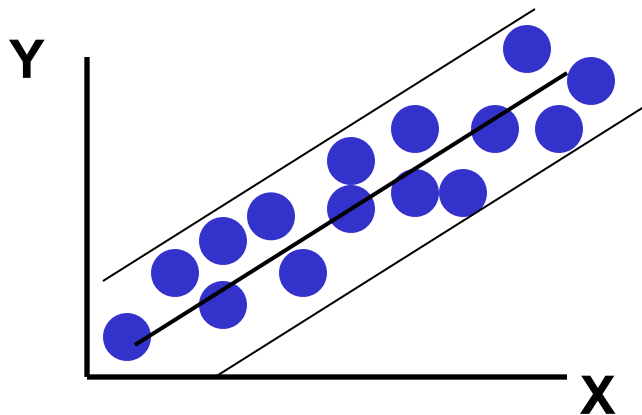


Curvilinear relationships

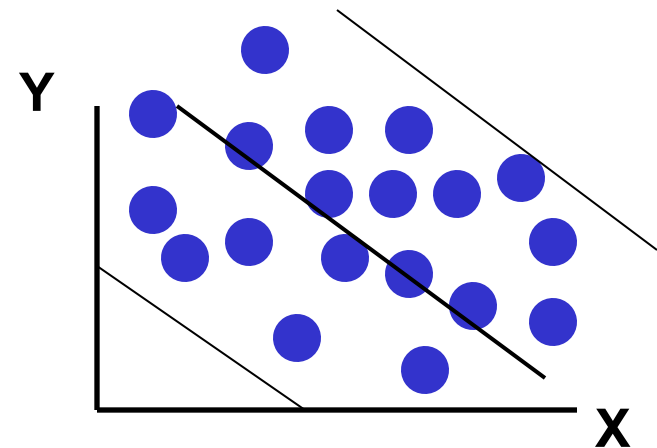
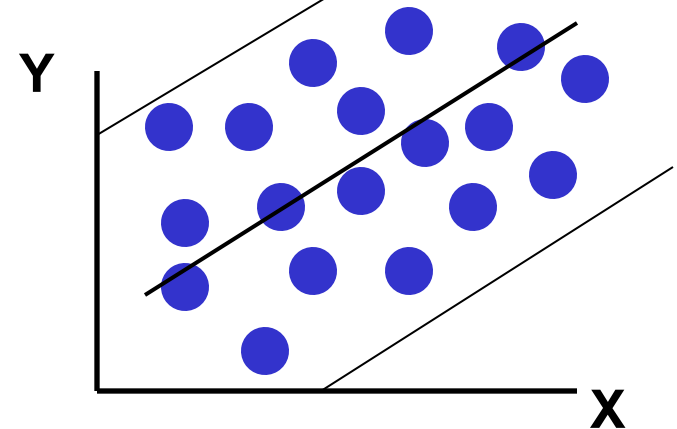


Linear Correlation

Strong relationships



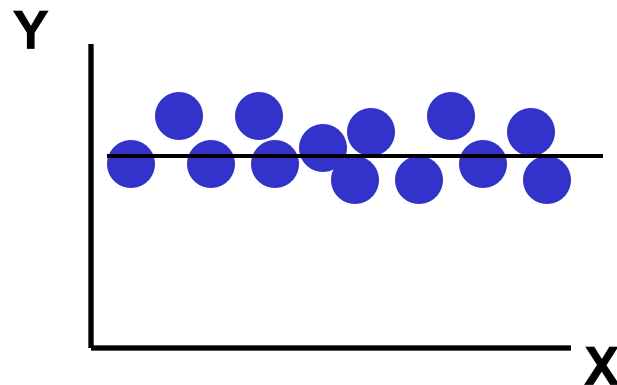
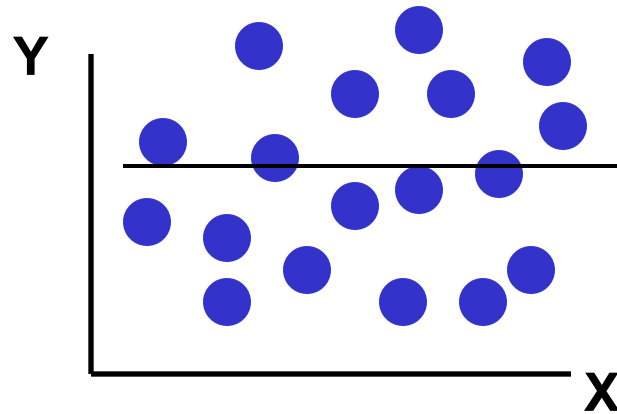
Weak relationships





Linear Correlation

No relationship





Calculating by hand...

$$\hat{r} = \frac{\text{covariance}(x, y)}{\sqrt{\text{var } x} \sqrt{\text{var } y}} = \frac{\frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n-1}}{\sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1}}}$$

Simpler calculation formula...

$$\hat{r} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1}}} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{SS_{xy}}{\sqrt{SS_x SS_y}}$$

**Numerator of
covariance**

$$\hat{r} = \frac{SS_{xy}}{\sqrt{SS_x SS_y}}$$

**Numerators of
variance**

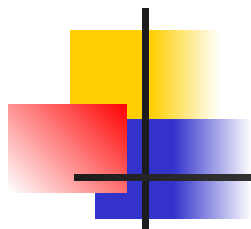


Distribution of the correlation coefficient:

$$SE(\hat{r}) = \sqrt{\frac{1 - r^2}{n - 2}}$$

The sample correlation coefficient follows a T-distribution with $n-2$ degrees of freedom (since you have to estimate the standard error).

*note, like a proportion, the variance of the correlation coefficient depends on the correlation coefficient itself □ substitute in estimated r



- In probability and statistics, Student's t-distribution (or simply the t-distribution) is a continuous probability distribution that generalizes the standard normal distribution. Like the latter, it is symmetric around zero and bell-shape.
- The Z distribution is a special case of the normal distribution with a mean of 0 and standard deviation of 1. The t-distribution is similar to the Z-distribution, but is sensitive to sample size and is used for small or moderate samples when the population standard deviation is unknown.



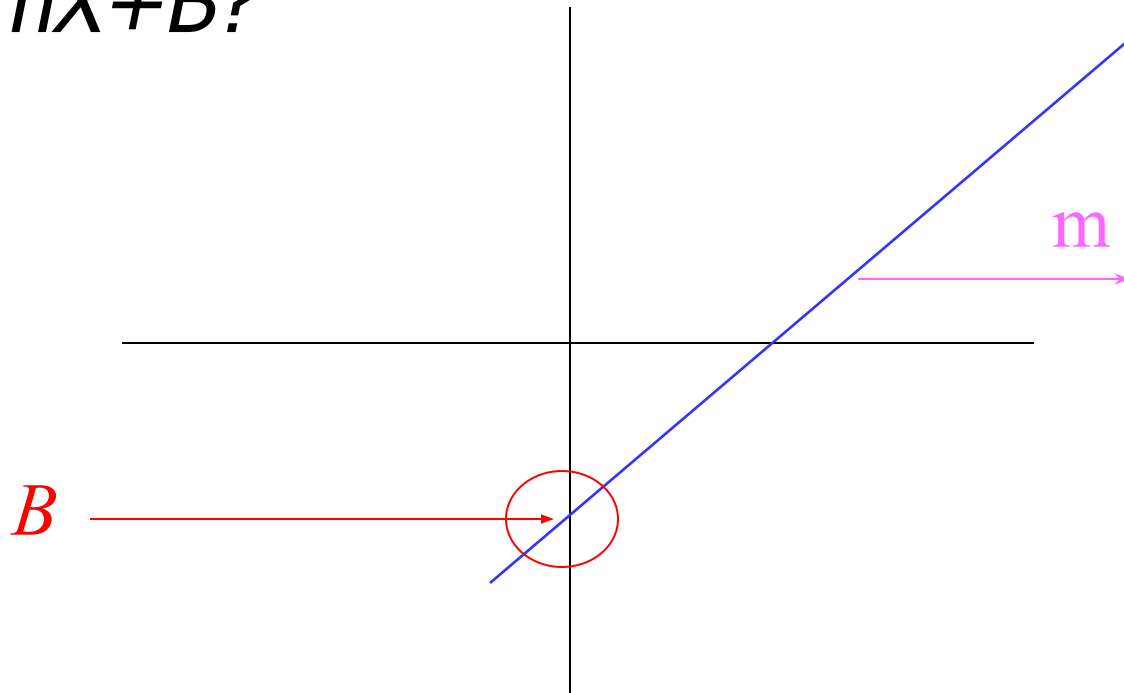
Linear regression

In correlation, the two variables are treated as equals. In regression, one variable is considered independent (=predictor) variable (X) and the other the dependent (=outcome) variable Y .



What is “Linear”?

- Remember this:
- $Y = mX + B$





What's Slope?

A slope of 2 means that every 1-unit change in X yields a 2-unit change in Y .



Prediction

If you know something about X , this knowledge helps you predict something about Y . (Sound familiar?...sound like conditional probabilities?)



Regression equation...

Expected value of y at a given level of x=

$$E(y_i / x_i) = \alpha + \beta x_i$$

Predicted value for an individual...



$$\hat{y}_i = \underbrace{\alpha + \beta * x_i}_{\text{Fixed -- exactly on the line}} + \boxed{\text{random error}_i}$$

Fixed –
exactly
on the
line

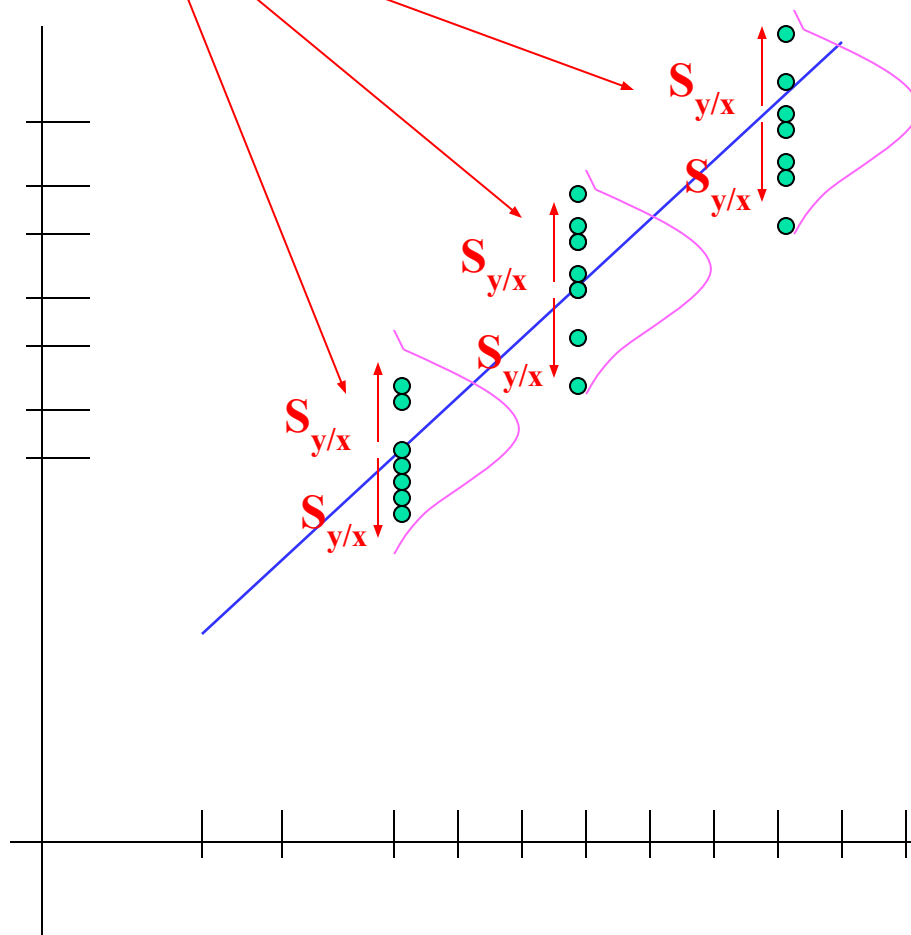
Follows a normal
distribution



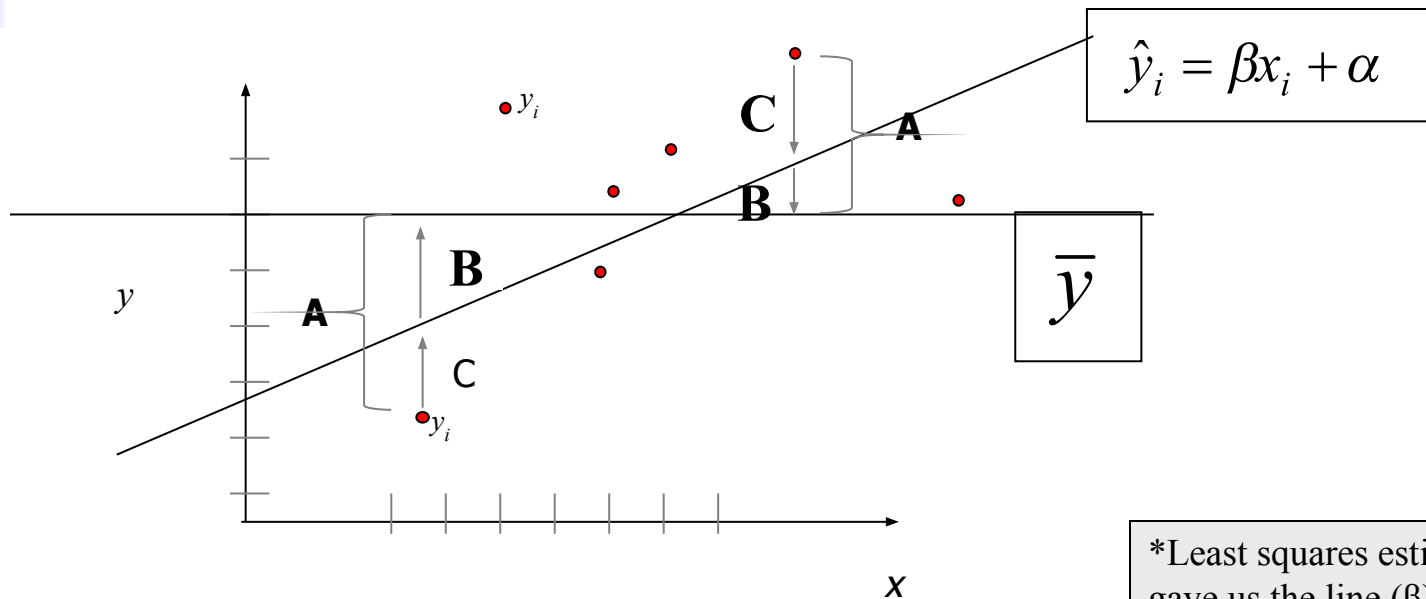
Assumptions (or the fine print)

- Linear regression assumes that...
 - 1. The relationship between X and Y is linear
 - 2. Y is distributed normally at each value of X
 - 3. The variance of Y at every value of X is the same (homogeneity of variances)
 - 4. The observations are independent

The standard error of Y given X is the average variability around the regression line at any given value of X. It is assumed to be equal at all values of X.



Regression Picture



*Least squares estimation gave us the line (β) that minimized C^2

$$\sum_{i=1}^n (y_i - \bar{y})^2 = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 + \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

A^2
 SS_{total}
Total squared distance of observations from naïve mean of y
Total variation

B^2
 SS_{reg}
 Distance from regression line to naïve mean of y
 Variability due to x (regression)

C^2
 SS_{residual}
 Variance around the regression line
 Additional variability not explained by x—what least squares method aims to minimize

$$R^2 = SS_{\text{reg}} / SS_{\text{total}}$$



Estimating the intercept and slope: least squares estimation

** Least Squares Estimation

A little calculus....

What are we trying to estimate? **β , the slope**, from

What's the constraint? We are trying to minimize the squared distance (hence the “least squares”) between the observations themselves and the predicted values, or (also called the “residuals”, or left-over unexplained variability)

$$\text{Difference}_i = y_i - (\beta x_i + \alpha) \quad \text{Difference}_i^2 = (y_i - (\beta x_i + \alpha))^2$$

Find the β that gives the minimum sum of the squared differences. How do you maximize a function? Take the derivative; set it equal to zero; and solve. Typical max/min problem from calculus....

$$\frac{d}{d\beta} \sum_{i=1}^n (y_i - (\beta x_i + \alpha))^2 = 2 \left(\sum_{i=1}^n (y_i - \beta x_i - \alpha)(-x_i) \right)$$

$$2 \left(\sum_{i=1}^n (-y_i x_i + \beta x_i^2 + \alpha x_i) \right) = 0 \dots$$

From here takes a little math trickery to solve for β ...



Resulting formulas...

Slope (beta coefficient) =

$$\hat{\beta} = \frac{Cov(x, y)}{Var(x)}$$

Intercept=

$$\text{Calculate: } \hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$$

Regression line always goes through the point: (\bar{x}, \bar{y})



Relationship with correlation

$$\hat{r} = \hat{\beta} \frac{SD_x}{SD_y}$$

In correlation, the two variables are treated as equals. In regression, one variable is considered independent (=predictor) variable (X) and the other the dependent (=outcome) variable Y .



Formula for the standard error of beta (you will not have to calculate by hand!):

$$s_{\hat{\beta}} = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n-2}} = \sqrt{\frac{s_{y/x}^2}{SS_x}}$$

$$\text{where } SS_x = \sum_{i=1}^n (x_i - \bar{x})^2$$

$$\text{and } \hat{y}_i = \hat{\alpha} + \hat{\beta}x_i$$

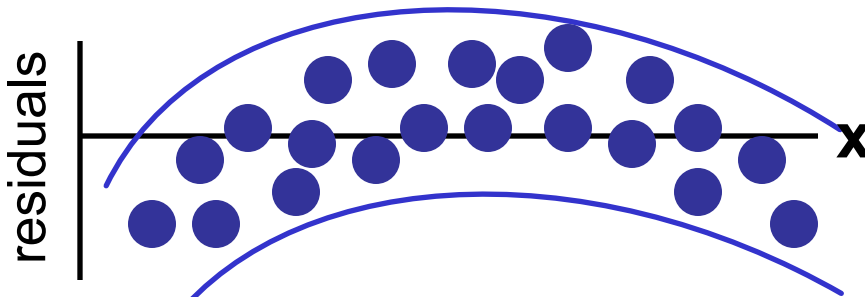
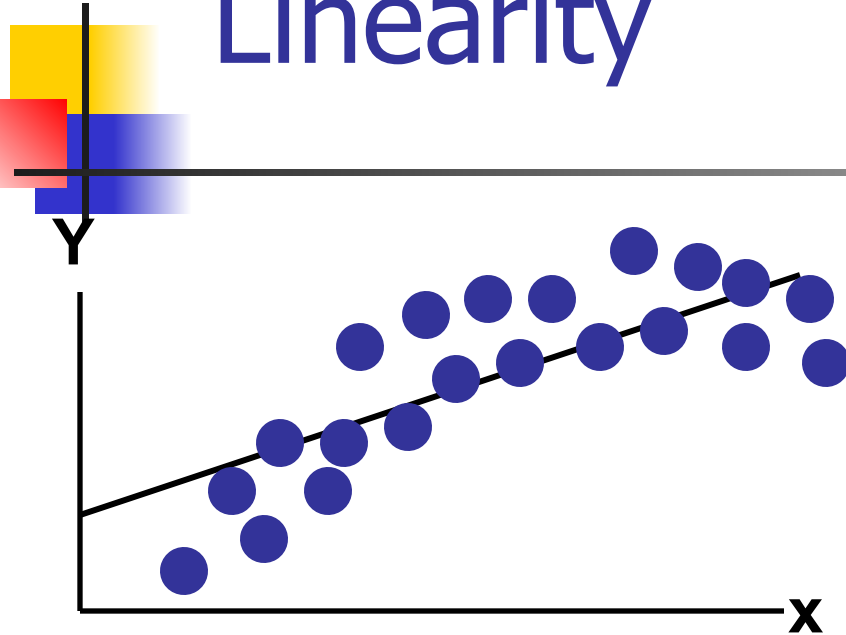


Residual Analysis: check assumptions

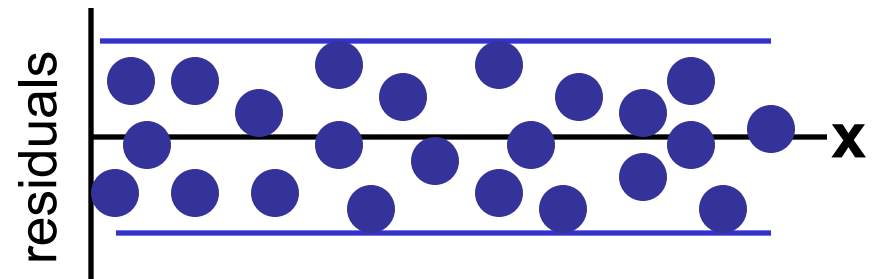
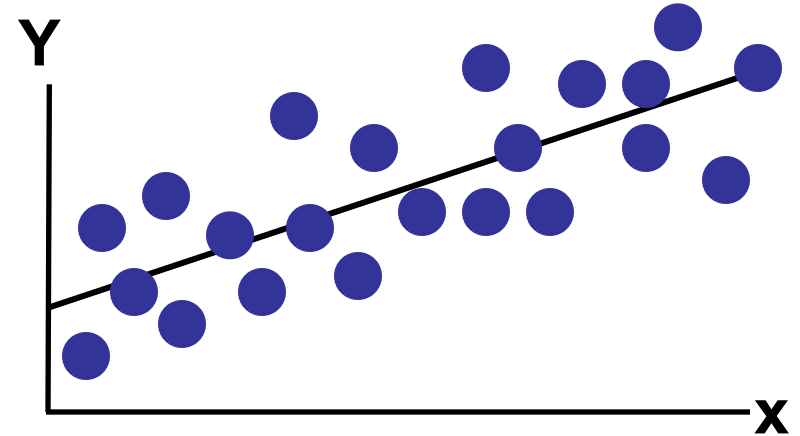
$$e_i = Y_i - \hat{Y}_i$$

- The residual for observation i , e_i , is the difference between its observed and predicted value
- Check the assumptions of regression by examining the residuals
 - Examine for linearity assumption
 - Examine for constant variance for all levels of X (homoscedasticity)
 - Evaluate normal distribution assumption
 - Evaluate independence assumption
- Graphical Analysis of Residuals
 - Can plot residuals vs. X

Residual Analysis for Linearity

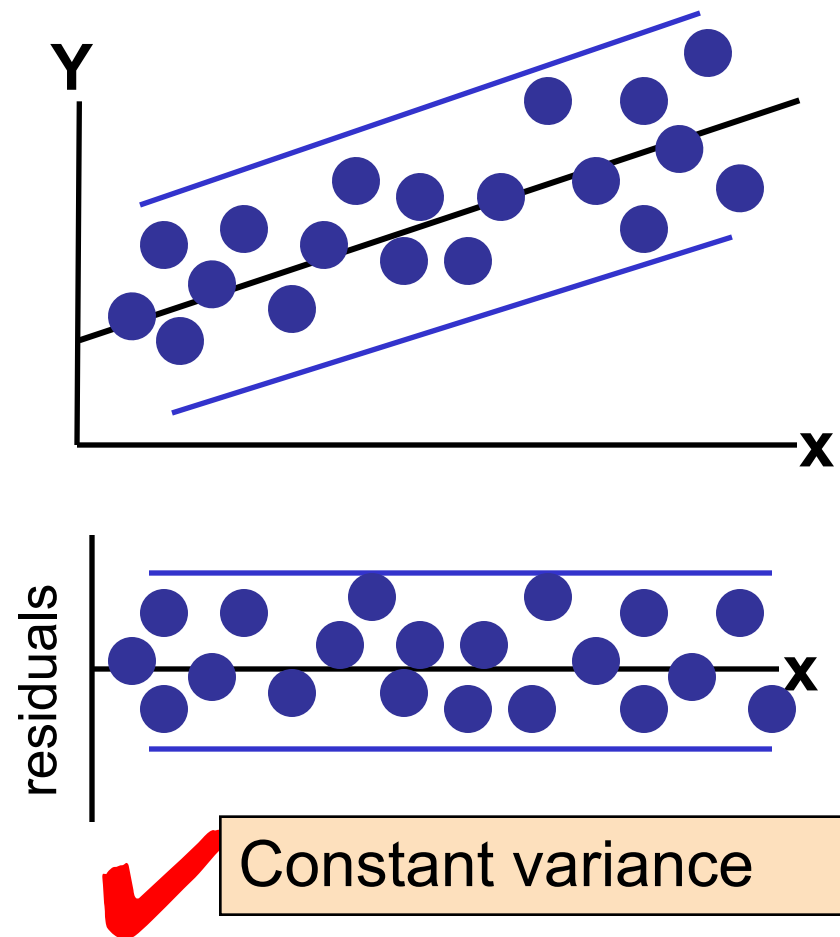
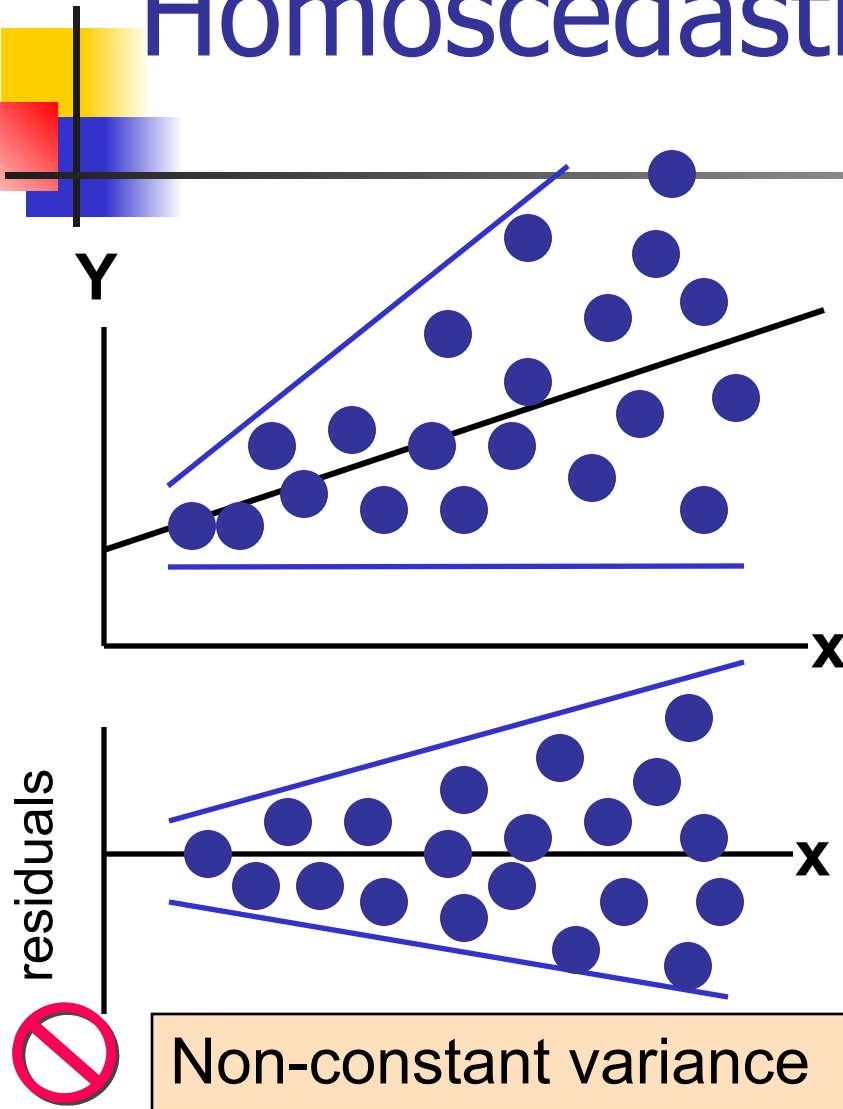


Not Linear

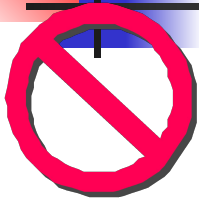


Linear

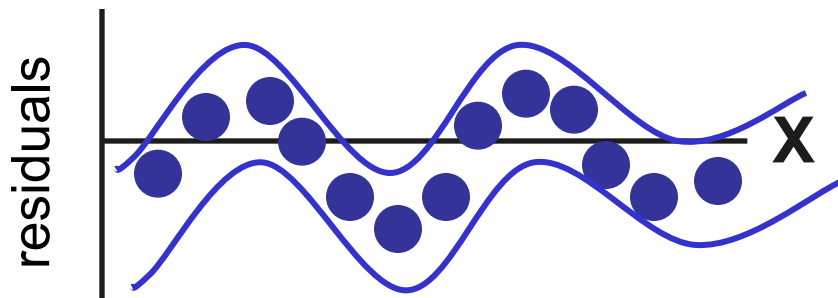
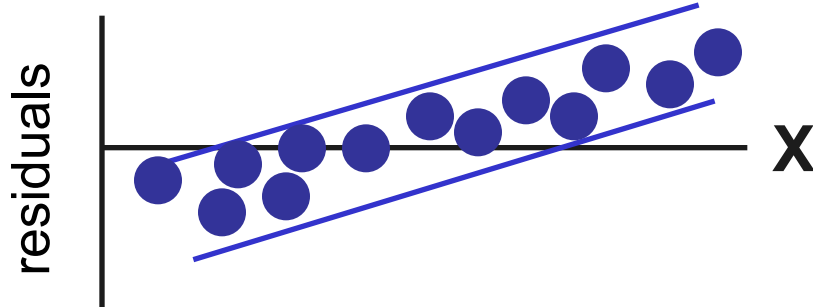
Residual Analysis for Homoscedasticity



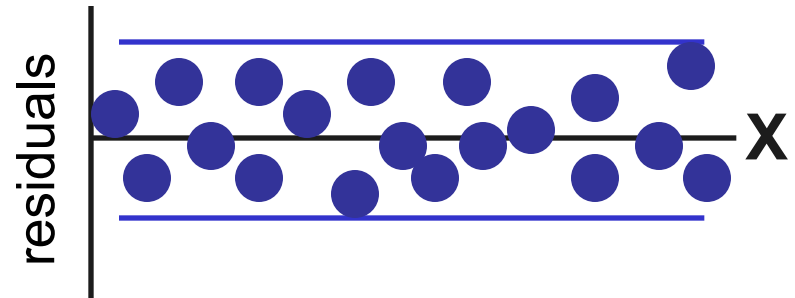
Residual Analysis for Independence



Not Independent



Independent





Other types of multivariate regression

- Multiple linear regression is for normally distributed outcomes
- Logistic regression is for binary outcomes
- Cox proportional hazards regression is used when time-to-event is the outcome

Common multivariate regression models.

Outcome (dependent variable)	Example outcome variable	Appropriate multivariate regression model	Example equation	What do the coefficients give you?
Continuous	Blood pressure	Linear regression	blood pressure (mmHg) = $\alpha + \beta_{\text{salt}} \cdot \text{salt consumption (tsp/day)} + \beta_{\text{age}} \cdot \text{age (years)} + \beta_{\text{smoker}} \cdot \text{ever smoker (yes=1/no=0)}$	slopes—tells you how much the outcome variable increases for every 1-unit increase in each predictor.
Binary	High blood pressure (yes/no)	Logistic regression	ln (odds of high blood pressure) = $\alpha + \beta_{\text{salt}} \cdot \text{salt consumption (tsp/day)} + \beta_{\text{age}} \cdot \text{age (years)} + \beta_{\text{smoker}} \cdot \text{ever smoker (yes=1/no=0)}$	odds ratios—tells you how much the odds of the outcome increase for every 1-unit increase in each predictor.
Time-to-event	Time-to- death	Cox regression	ln (rate of death) = $\alpha + \beta_{\text{salt}} \cdot \text{salt consumption (tsp/day)} + \beta_{\text{age}} \cdot \text{age (years)} + \beta_{\text{smoker}} \cdot \text{ever smoker (yes=1/no=0)}$	hazard ratios—tells you how much the rate of the outcome increases for every 1-unit increase in each predictor.



Multivariate regression pitfalls

- **Multi-collinearity**
- **Residual confounding**
- **Overfitting**



Multicollinearity

- **Multicollinearity** arises when two variables that measure the same thing or similar things (e.g., weight and BMI) are both included in a multiple regression model; they will, in effect, cancel each other out and generally destroy your model.
- Model building and diagnostics are tricky business!



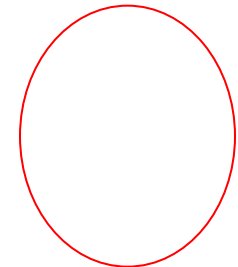
Overfitting

- In multivariate modeling, you can get highly significant but meaningless results if you put too many predictors in the model.
- The model is fit perfectly to the quirks of your particular sample, but has no predictive ability in a new sample.

Overfitting: class data example

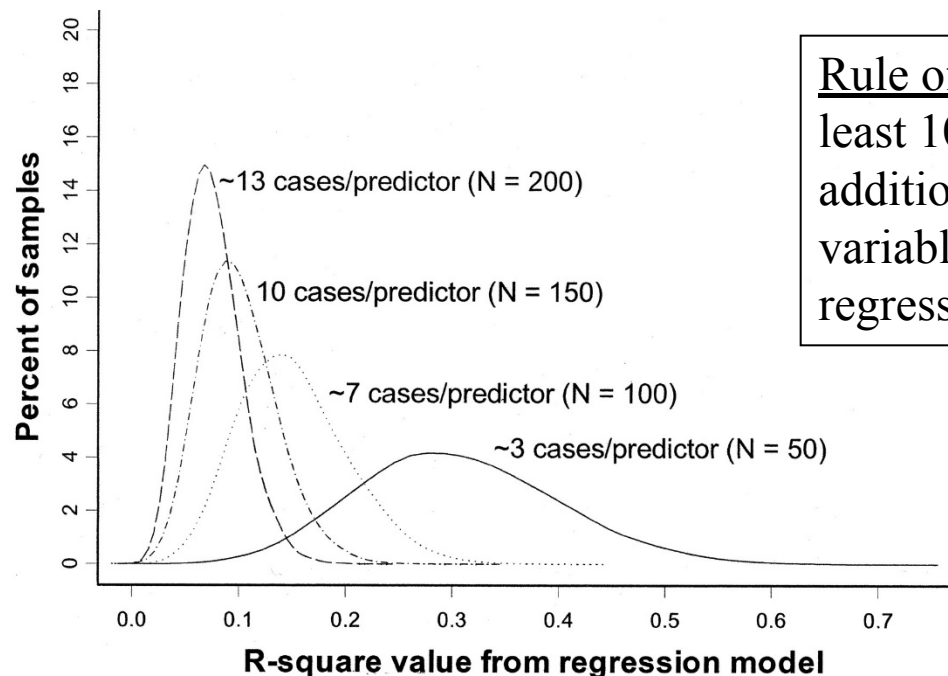
- I asked SAS to automatically find predictors of optimism in our class dataset. Here's the resulting linear regression model:

Variable	Parameter Estimate	Standard Error	Type II SS	F Value	Pr > F
Intercept	11.80175	2.98341	11.96067	15.65	0.0019
exercise	-0.29106	0.09798	6.74569	8.83	0.0117
sleep	-1.91592	0.39494	17.98818	23.53	0.0004
obama	1.73993	0.24352	39.01944	51.05	<.0001
Clinton	-0.83128	0.17066	18.13489	23.73	0.0004
mathLove	0.45653	0.10668	13.99925	18.32	0.0011



Exercise, sleep, and high ratings for Clinton are negatively related to optimism (*highly significant!*) and high ratings for Obama and high love of math are positively related to optimism (*highly significant!*).

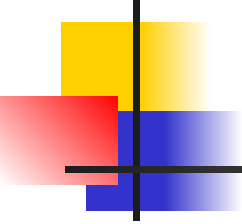
Overfitting



Rule of thumb: You need at least 10 subjects for each additional predictor variable in the multivariate regression model.

Pure noise variables still produce good R^2 values if the model is overfitted. The distribution of R^2 values from a series of simulated regression models containing only noise variables.

(Figure 1 from: Babyak, MA. What You See May Not Be What You Get: A Brief, Nontechnical Introduction to Overfitting in Regression-Type Models. *Psychosomatic Medicine* 66:411-421 (2004).)

- 
-
- R-Squared values range from 0 to 1. An R-Squared value of 0 means that the model explains or predicts 0% of the relationship between the dependent and independent variables. A value of 1 indicates that the model predicts 100% of the relationship, and a value of 0.5 indicates that the model predicts 50%, and so on



K-Nearest Neighbor



Different Learning Methods

- Eager Learning
 - Explicit description of target function on the whole training set
- Instance-based Learning
 - Learning=storing all training instances
 - Classification=assigning target function to a new instance
 - Referred to as “Lazy” learning



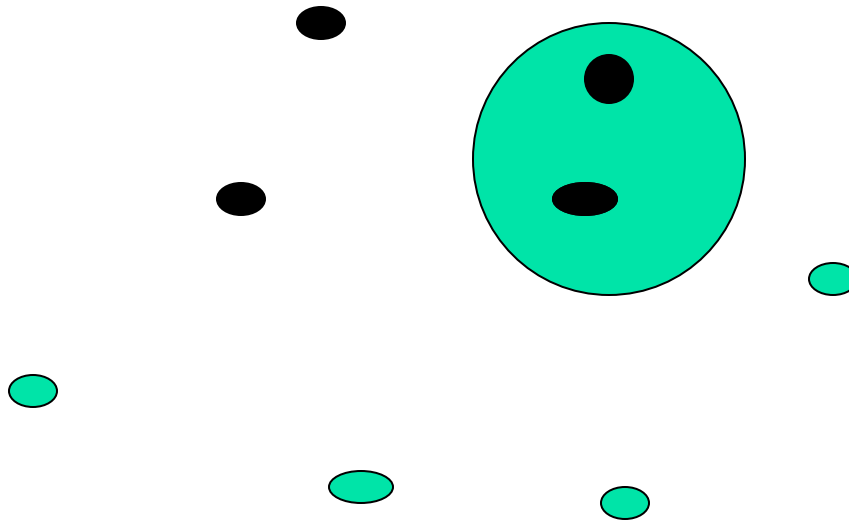
K-Nearest Neighbor

- Features

- All instances correspond to points in an n-dimensional Euclidean space
- Classification is delayed till a new instance arrives
- Classification done by comparing feature vectors of the different points
- Target function may be discrete or real-valued

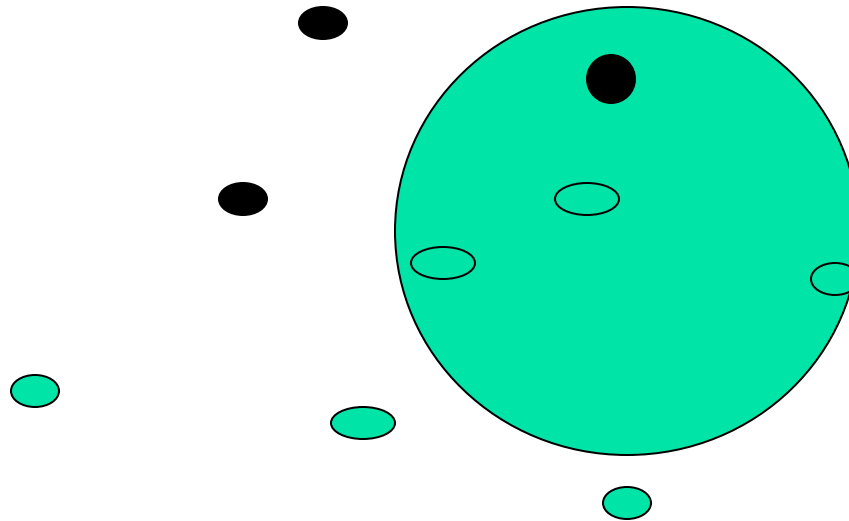


1-Nearest Neighbor





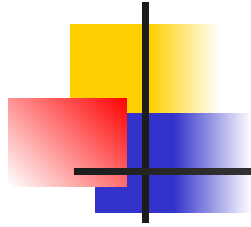
3-Nearest Neighbor





K-Nearest Neighbor

- An arbitrary instance is represented by $(a_1(x), a_2(x), a_3(x), \dots, a_n(x))$
 - $a_i(x)$ denotes features
- Euclidean distance between two instances
$$d(x_i, x_j) = \sqrt{\sum_{r=1}^n (a_r(x_i) - a_r(x_j))^2}$$
- Continuous valued target function
 - mean value of the k nearest training examples



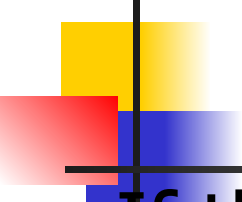
- **K-nearest neighbours uses the local neighborhood to obtain a prediction**
- **The K memorized examples more similar to the one that is being classified are retrieved**
- **A distance function is needed to compare the examples similarity**

Euclidean :

$$d(x, y) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

Manhattan / city - block :

$$d(x, y) = \sum_{i=1}^m |x_i - y_i|$$

- 
- If the ranges of the features differ, features with bigger values will dominate decision
 - In general feature values are normalized prior to distance calculation

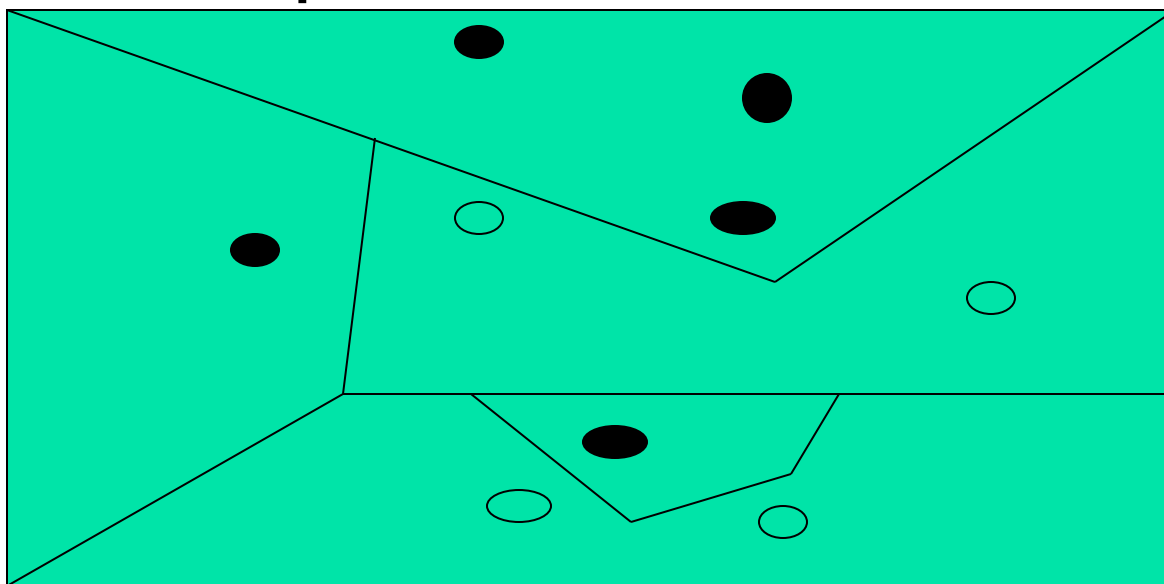
$$X_s = \frac{X - \text{mean}}{s.d.}$$

$$X_s = \frac{X - \text{mean}}{\text{max} - \text{min}}$$

$$X_s = \frac{X - \text{min}}{\text{max} - \text{min}}$$

Voronoi Diagram

- Decision surface formed by the training examples





Distance-Weighted Nearest Neighbor Algorithm

- Assign weights to the neighbors based on their 'distance' from the query point
 - Weight 'may' be inverse square of the distances
- All training points may influence a particular instance
 - Shepard's method



Remarks

- + Highly effective inductive inference method for noisy training data and complex target functions
- + Target function for a whole space may be described as a combination of less complex local approximations
- + Learning is very simple
- Classification is time consuming



Nearest-Neighbor Classifiers: Issues

- **The value of k , the number of nearest neighbors to retrieve**

- **Choice of Distance Metric to compute distance between records**

- **Computational complexity**

- **Size of training set**

- **Dimension of data**

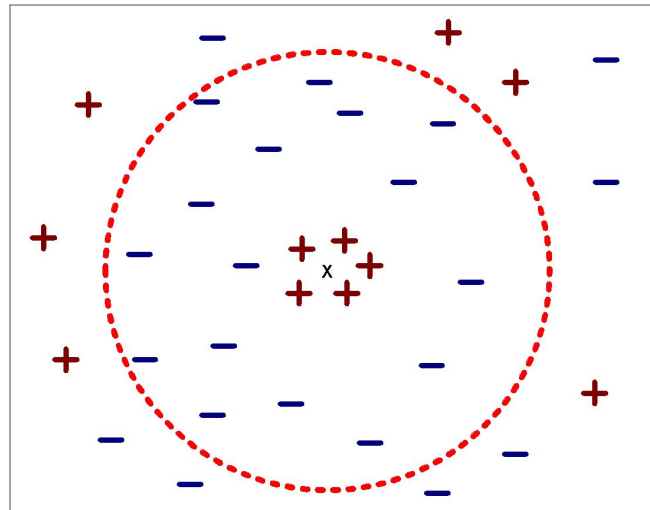
Value of K

- Choosing the value of k:
 - If k is too small, sensitive to noise points
 - If k is too large, neighborhood may include points from other classes

Rule of thumb:

$K = \sqrt{N}$

N: number of training points



Distance Metrics

Minkowsky:

$$D(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^m |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m |x_i - y_i|$$

Camberra:

$$D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\mathbf{x}, \mathbf{y}) = \max_{i=1}^m |x_i - y_i|$$

Quadratic:

$$D(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T Q (\mathbf{x} - \mathbf{y}) = \sum_{j=1}^m \left(\sum_{i=1}^m (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\mathbf{x}, \mathbf{y}) = [\det V]^{1/m} (\mathbf{x} - \mathbf{y})^T V^{-1} (\mathbf{x} - \mathbf{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occurring in the training set instances 1.. n .

Correlation:

$$D(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^m (x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^m (x_i - \bar{x}_i)^2 \sum_{i=1}^m (y_i - \bar{y}_i)^2}}$$

$\bar{x}_i = \bar{y}_i$ and is the average value for attribute i occurring in the training set.

Chi-square:

$$D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m \frac{1}{sum_i} \left(\frac{x_i}{size_x} - \frac{y_i}{size_y} \right)^2$$

sum_i is the sum of all values for attribute i occurring in the training set, and $size_x$ is the sum of all values in the vector \mathbf{x} .

Kendall's Rank Correlation:

$$D(\mathbf{x}, \mathbf{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^m \sum_{j=1}^{i-1} \text{sign}(x_i - x_j) \text{sign}(y_i - y_j)$$

$\text{sign}(x) = -1, 0$ or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions.
(\mathbf{x} and \mathbf{y} are vectors of m attribute values).

Distance Measure: Scale Effects



- Different features may have different measurement scales
 - E.g., patient weight in kg (range [50,200]) vs. blood protein values in ng/dL (range [-3,3])
- Consequences
 - Patient weight will have a much greater influence on the distance between samples
 - May bias the performance of the classifier



Nearest Neighbour : Computational Complexity

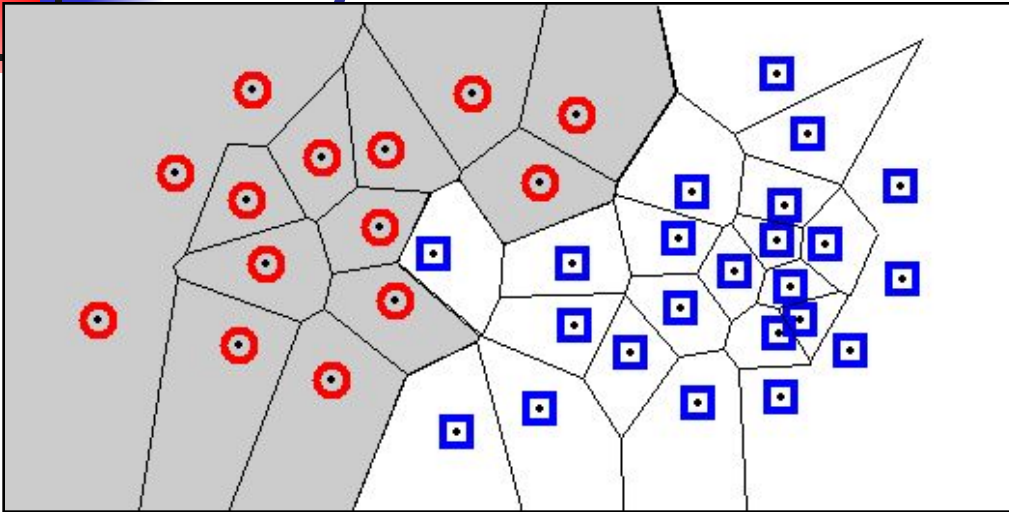
- Expensive
 - To determine the nearest neighbour of a query point q , must compute the distance to all N training examples
 - + Pre-sort training examples into fast data structures (kd-trees)
 - + Compute only an approximate distance
 - + Remove redundant data (condensing)
- Storage Requirements
 - Must store all training data **P**
 - + Remove redundant data (condensing)
 - Pre-sorting often increases the storage requirements
- High Dimensional Data
 - “Curse of Dimensionality”
 - Required amount of training data increases exponentially with dimension
 - Computational cost also increases dramatically
 - Partitioning techniques degrade to linear search in high dimension



Reduction in Computational Complexity

- Reduce size of training set
 - Condensation, editing
- Use geometric data structure for high dimensional search

Condensation: Decision Regions



Each cell contains one sample, and every location within the cell is closer to that sample than to any other sample.

A Voronoi diagram divides the space into such cells.

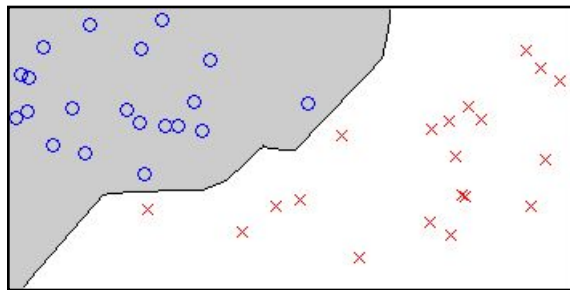
Every query point will be assigned the classification of the sample within that cell. The *decision boundary* separates the class regions based on the 1-NN decision rule.

Knowledge of this boundary is sufficient to classify new points.

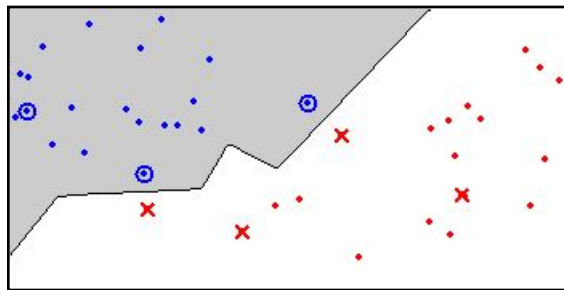
The boundary itself is rarely computed; many algorithms seek to retain only those points necessary to generate an identical boundary.

Condensing

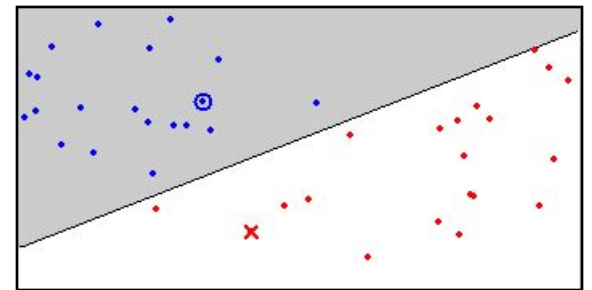
- Aim is to reduce the number of training samples
- Retain only the samples that are needed to define the decision boundary
- Decision Boundary Consistent – a subset whose nearest neighbour decision boundary is identical to the boundary of the entire training set
- Minimum Consistent Set – the smallest subset of the training data that correctly classifies all of the original training data



Original data



Condensed data

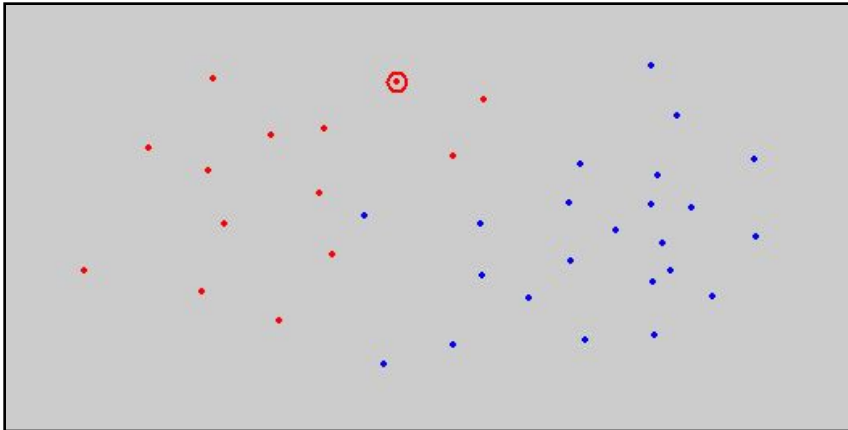


Minimum Consistent Set

Condensing

■ Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single (or K) training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers

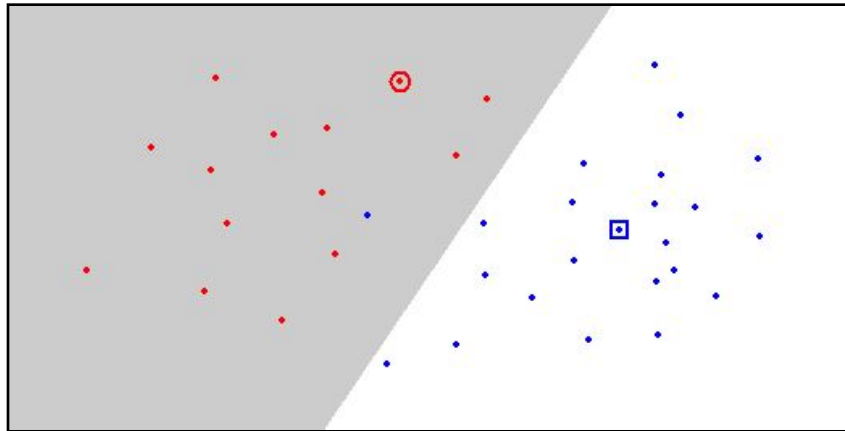


- Incremental
- Order dependent
- Neither minimal nor decision boundary consistent
- $O(n^3)$ for brute-force method

Condensing

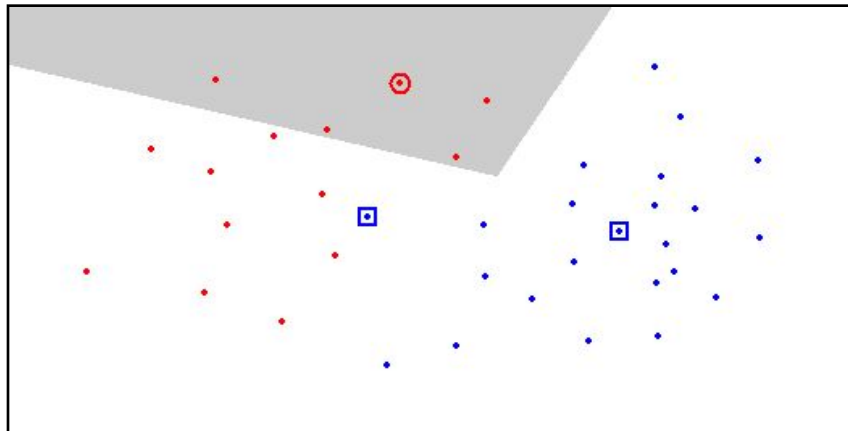
Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full



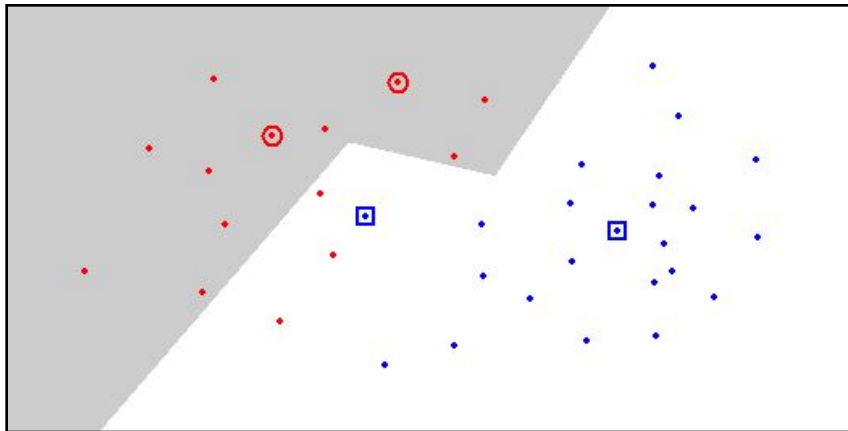
Condensing

Condensed Nearest Neighbour (CNN)



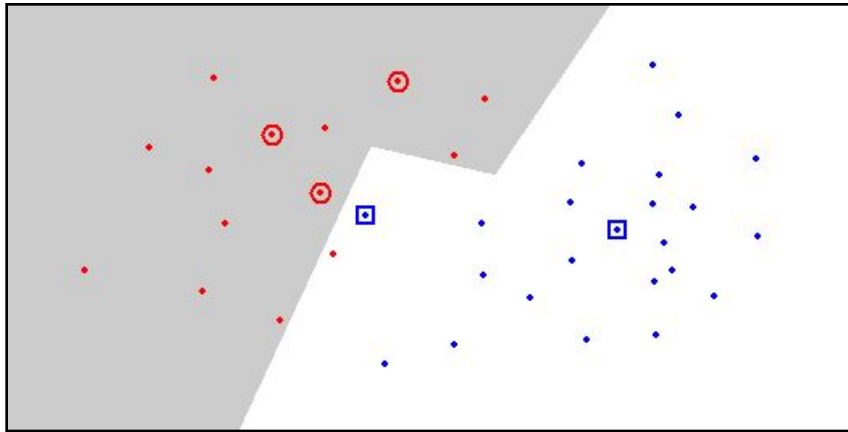
Condensing

Condensed Nearest Neighbour (CNN)



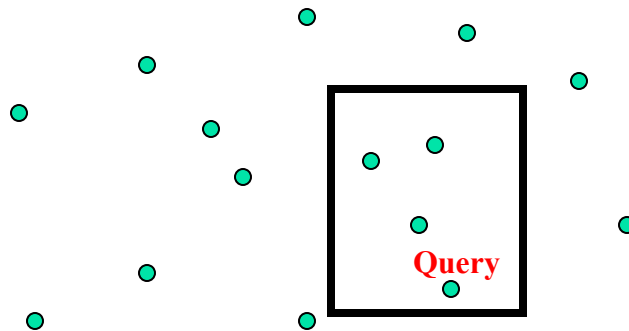
Condensing

Condensed Nearest Neighbour (CNN)

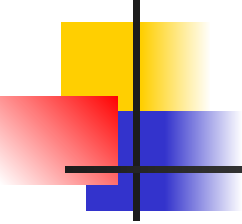


High dimensional search

- Given a point set and a nearest neighbor query point
- Find the points enclosed in a rectangle (range) around the query
- Perform linear search for nearest neighbor only in the rectangle



KNN: Alternate Terminologies

- 
-
- Instance Based Learning
 - Lazy Learning
 - Case Based Reasoning
 - Exemplar Based Learning



Classifier Accuracy



How it can be measured?

Holdout Method (Random Sub sampling)

K-fold Cross Validation

Bootstrapping



How we can improve classifier Accuracy?

Bagging

Boosting



Is accuracy enough to judge a classifier?



Predictor Error Measures

- Measure predictor accuracy: measure how far off the predicted value is from the actual known value
 - **Loss function:** measures the error between y_i and the predicted value y_i'
 - Absolute error: $|y_i - y_i'|$
 - Squared error: $(y_i - y_i')^2$
 - Test error (generalization error): the average loss over the test set
 - Mean absolute error: $\frac{\sum_{i=1}^d |y_i - y_i'|}{d}$ Mean squared error: $\frac{\sum_{i=1}^d (y_i - y_i')^2}{d}$
 - Relative absolute error: $\frac{\sum_{i=1}^d |y_i - y_i'|}{\sum_{i=1}^d |y_i - \bar{y}|}$ Relative squared error: $\frac{\sum_{i=1}^d (y_i - y_i')^2}{\sum_{i=1}^d (y_i - \bar{y})^2}$
- The mean squared-error exaggerates the presence of outliers
- Popularly use (square) root mean-square error, similarly, root relative squared error

Evaluating the Accuracy of a Classifier or Predictor (I)

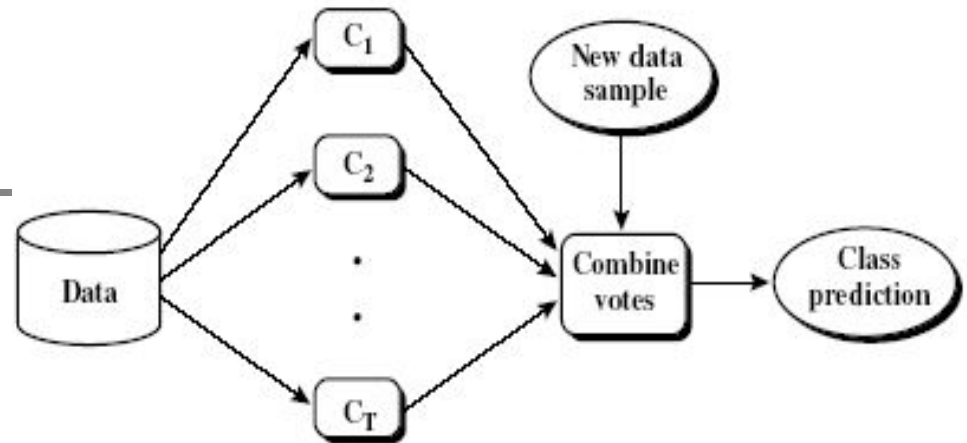
- Holdout method
 - Given data is randomly partitioned into two independent sets
 - Training set (e.g., 2/3) for model construction
 - Test set (e.g., 1/3) for accuracy estimation
 - Random sampling: a variation of holdout
 - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- Cross-validation (k -fold, where $k = 10$ is most popular)
 - Randomly partition the data into k *mutually exclusive* subsets, each approximately equal size
 - At i -th iteration, use D_i as test set and others as training set
 - Leave-one-out: k folds where $k = \#$ of tuples, for small sized data
 - Stratified cross-validation: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Evaluating the Accuracy of a Classifier or Predictor (II)

- Bootstrap
 - Works well with small data sets
 - Samples the given training tuples uniformly *with replacement*
 - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 bootstrap**
 - Suppose we are given a data set of d tuples. The data set is sampled d times, with replacement, resulting in a training set of d samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data will end up in the bootstrap, and the remaining 36.8% will form the test set (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)
 - Repeat the sampling procedure k times, overall accuracy of the model:

$$acc(M) = \sum_{i=1}^k (0.632 \times acc(M_i)_{test_set} + 0.368 \times acc(M_i)_{train_set})$$

Ensemble Methods: Increasing the Accuracy




- Ensemble methods
 - Use a combination of models to increase accuracy
 - Combine a series of k learned models, M_1, M_2, \dots, M_k , with the aim of creating an improved model M^*
- Popular ensemble methods
 - Bagging: averaging the prediction over a collection of classifiers (e.g. Random Forest)
 - Boosting: weighted vote with a collection of classifiers
 - Ensemble: combining a set of heterogeneous classifiers



Bagging: Bootstrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
 - Given a set D of d tuples, at each iteration i , a training set D_i of d tuples is sampled with replacement from D (i.e., bootstrap)
 - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample \mathbf{X}
 - Each classifier M_i returns its class prediction
 - The bagged classifier M^* counts the votes and assigns the class with the most votes to \mathbf{X}
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
 - Often significant better than a single classifier derived from D
 - For noise data: not considerably worse, more robust
 - Proved improved accuracy in prediction

Boosting

- 
-
- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
 - How boosting works?
 - Weights are assigned to each training tuple
 - A series of k classifiers is iteratively learned
 - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1} , to pay more attention to the training tuples that were misclassified by M_i
 - The final M^* combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
 - The boosting algorithm can be extended for the prediction of continuous values
 - Comparing with bagging: boosting tends to achieve greater accuracy, but it also risks overfitting the model to misclassified data



Adaboost (Freund and Schapire, 1997)

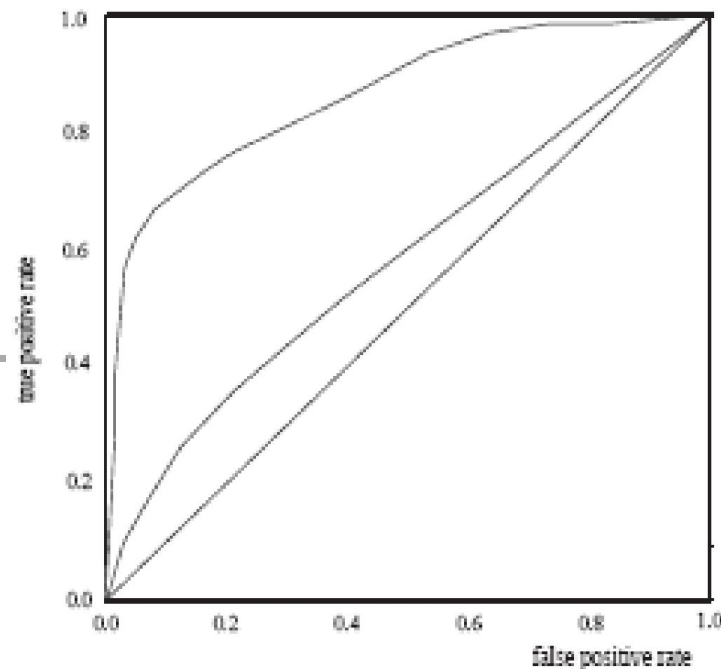
- Given a set of d class-labeled tuples, $(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_d, y_d)$
- Initially, all the weights of tuples are set the same ($1/d$)
- Generate k classifiers in k rounds. At round i ,
 - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
 - Each tuple's chance of being selected is based on its weight
 - A classification model M_i is derived from D_i
 - Its error rate is calculated using D_i as a test set
 - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: $err(\mathbf{X}_j)$ is the misclassification error of tuple \mathbf{X}_j . Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_j^d w_j \times err(\mathbf{X}_j)$$

- The weight of classifier M_i 's vote is $\log \frac{1 - error(M_i)}{error(M_i)}$

Model Selection: ROC Curves

- ROC (Receiver Operating Characteristics) curves: for visual comparison of classification models
- Originated from signal detection theory
- Shows the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve is a measure of the accuracy of the model
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model



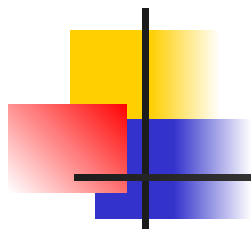
- **Vertical axis represents the true positive rate**
- **Horizontal axis rep. the false positive rate**
- **The plot also shows a diagonal line**
- **A model with perfect accuracy will have an area**

of 1.0



Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
- Typical methods for imbalance data in 2-class classification:
 - **Oversampling**: re-sampling of data from positive class
 - **Under-sampling**: randomly eliminate tuples from negative class
 - **Threshold-moving**: moves the decision threshold, t , so that the rare class tuples are easier to classify, and hence, less chance of false negative errors
 - Ensemble techniques: Ensemble multiple classifiers introduced above



- ***Feedback: manish@iiiita.ac.in***
- ***I acknowledge all the authors and websites whose content was part of my lecture.***
- ***Thanks***