CSCI446/946 Big Data Analytics

Week 5 – Lecture: Classification

School of Computing and Information Technology
University of Wollongong Australia

Spring 2024

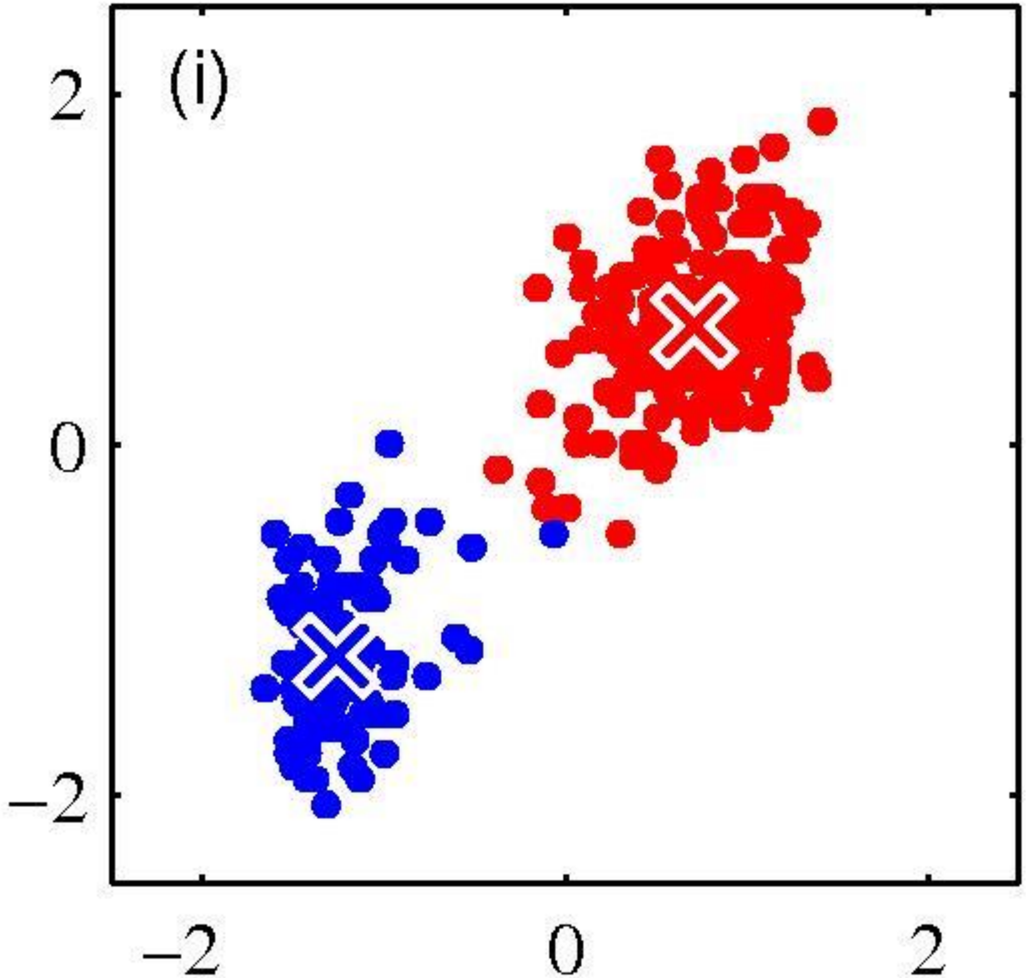
Content

- Brief Recap
  - Clustering Analysis
    - K-means, DBSCAN, SOM
- Classification
  - Overview
  - K-Nearest Neighbor (KNN)
  - Multi-Layers Perceptron (MLP)
  - Decision Tree (DT)
  - Naïve Bayesian Classifier
- Diagnostics and Performance Indicators

Content

- Brief Recap
  - Clustering Analysis
    - K-means, DBSCAN, SOM
- Classification
  - Overview
  - K-Nearest Neighbor (KNN)
  - Multi-Layers Perceptron (MLP)
  - Decision Tree (DT)
  - Naïve Bayesian Classifier
- Diagnostics and Performance Indicators

K-means Clustering



K-means Clustering

- Application to image processing

K-means Clustering

- Application to image processing

Original

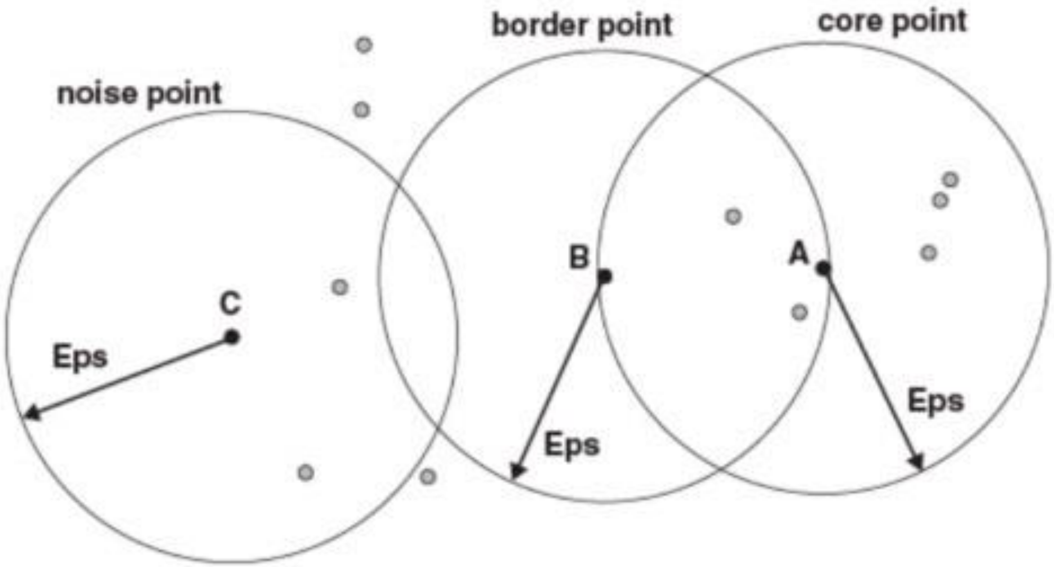
K=2

K=3

K=10

DBScan

- Given a density threshold ($MinPts$) and a radius (Eps), the points in a dataset are classified into three types: core point, border point, and noise point.
  – Core points: $Point\ whose\ density\ >=\ MinPts$
  – Core points are in the interior of a density-based cluster.

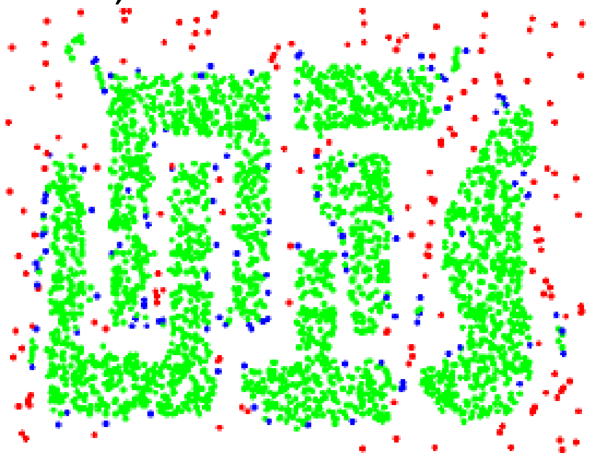


Example: If $MinPts\ =\ 6$ then A is a core point because its density = 7 (7>6)

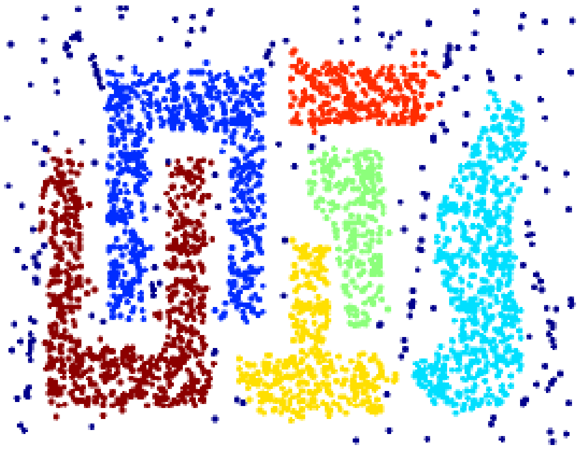
DBScan Example

Original Points

Eps = 10, MinPts = 4



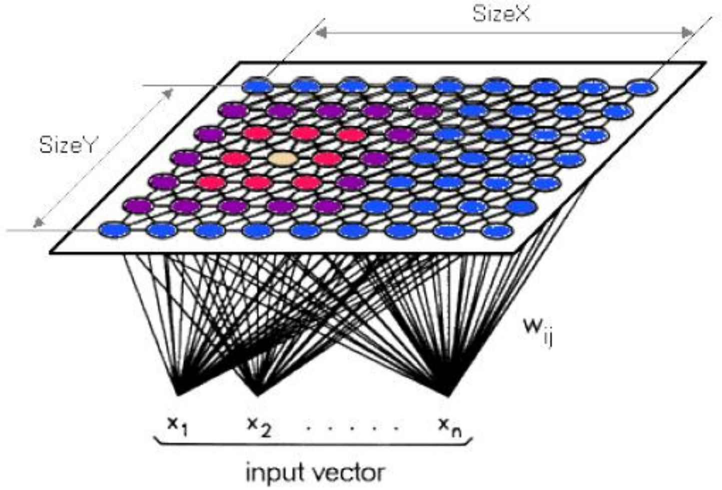
Mark core, border and noise points



Mark connected core points

Self-Organizing Maps

- Self-organizing maps have two layers:
  - An input layer and
  - An output layer called the feature map.
- The feature map consists of neurons.
  - organized on a regular grid.
  - Unlike other ANN types, the neuro in a SOM don't have an activation function.



- Each neuron in a SOM is assigned a weight vector with the same dimensionality as the input space.

Self-Organizing Maps

- SOMs are an excellent choice for data visualization
  - Dimension reduction techniques
- Why use Self-Organizing Maps (SOMs) in BDA?
  - Topology preservation (unlike PCA)
  - Able to deal with new data & missing values (unlike t-SNE)
- When not to use SOMs in BDA:
  - When the data is very sparse
  - When cardinality (limited resolution) of the map is a problem.

Content

- Brief Recap
  - Clustering Analysis
    - K-means, DBSCAN, SOM
- Classification
  - Overview
  - K-Nearest Neighbor (KNN)
  - Multi-Layers Perceptron (MLP)
  - Decision Tree (DT)
  - Naïve Bayesian Classifier
- Diagnostics and Performance Indicators

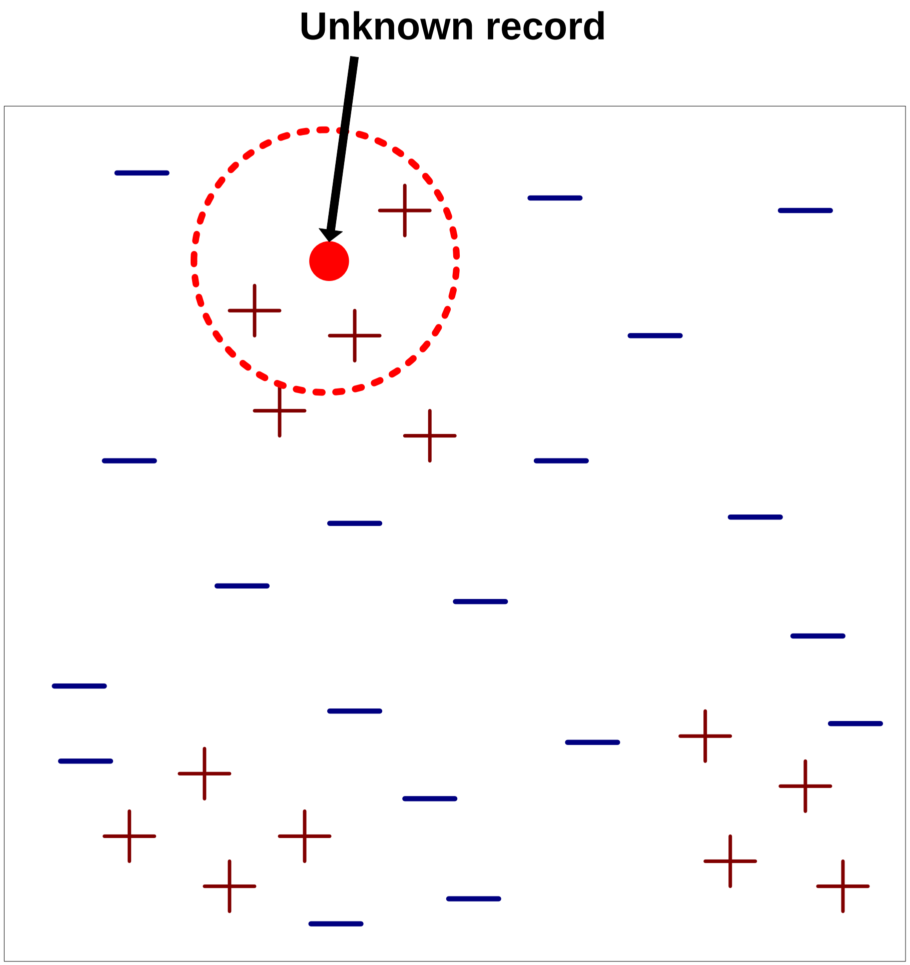
Overview of Classification

- Classification is a fundamental learning method that appears in applications related to data mining
- The primary task performed by classifiers is to assign class labels to new observations
  - Sets: training, (validation), testing
- Classification methods are supervised
  - Start with a training set of labelled observations
  - Predict the outcome for new observations

Overview of Classification

- Example of classifiers:
  - K-nearest neighbour (KNN): model free classifier
  - Neural Networks (NN): Massive parallel nonlinear parametric methods
  - Decision Tree and Random Forests: Makes explanatory if-then decisions
  - Naïve Bayes (NB) Classifier: Probabilistic methods
  - Logistic regression: Linear method (LR)
  - Support Vector Machines (SVM): non-parametric classifiers.
  - …

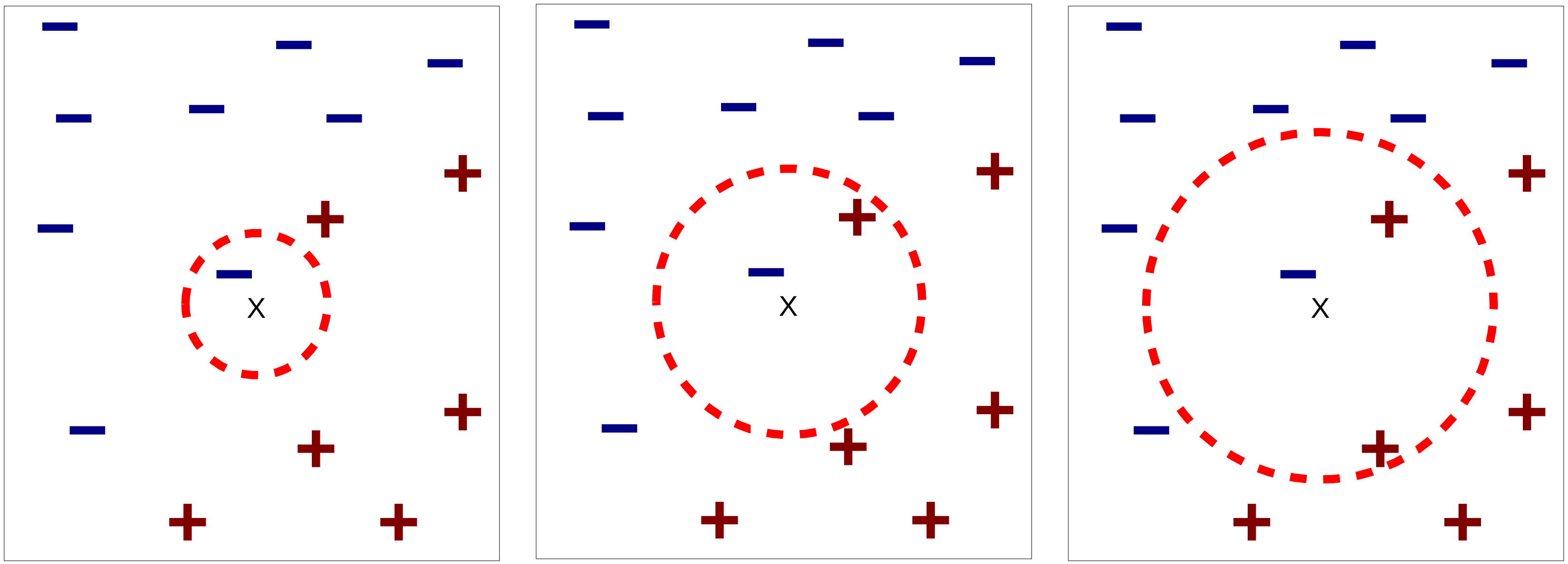
Nearest-Neighbor Classifiers



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Nearest Neighbor Classification

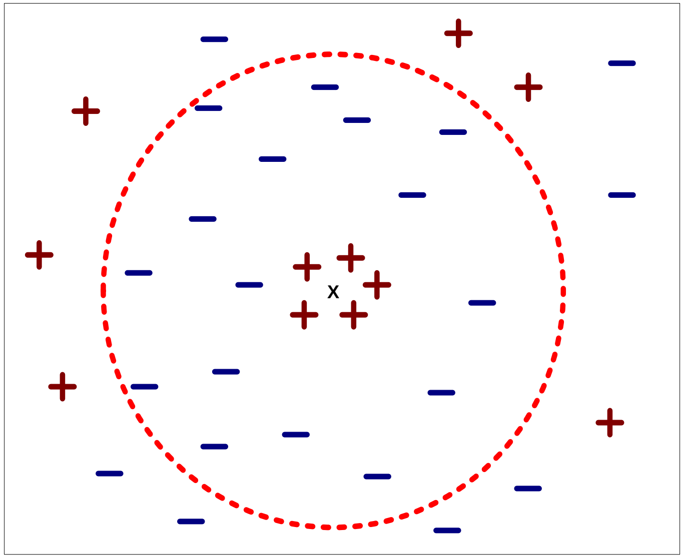
- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

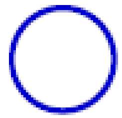
- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

Nearest Neighbor Classification…

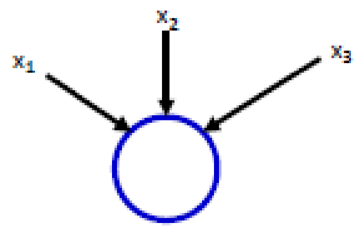
- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes
  - Computational cost often increases when k increases



Neural Networks - MLP

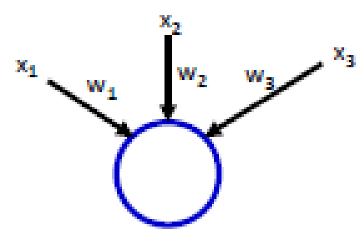
1. A neuron: 

2. Receives inputs:



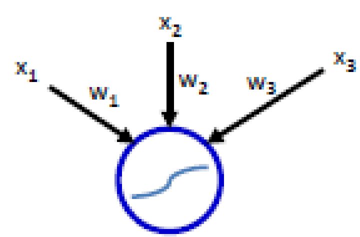
$$\sum x_i$$

3. Incoming links are weighted: (Control amount of information that is propagated)



$$\sum_i w_i x_i$$

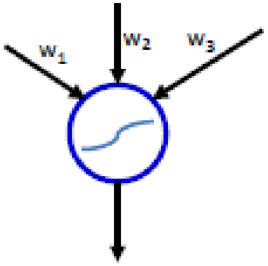
4. Processes inputs (activation function):
(Biological neurons are spiking. Activation function simulates the total amount of energy of the spikes)



$$f(\sum_i w_i x_i)$$

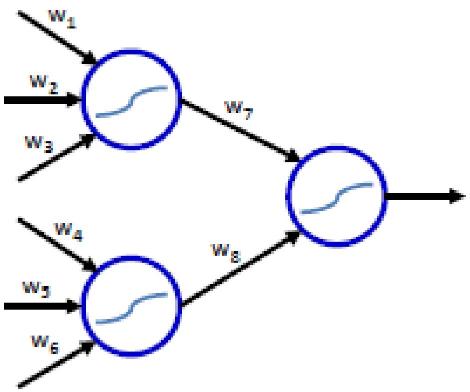
Neural Networks - MLP

5. Produces an output:



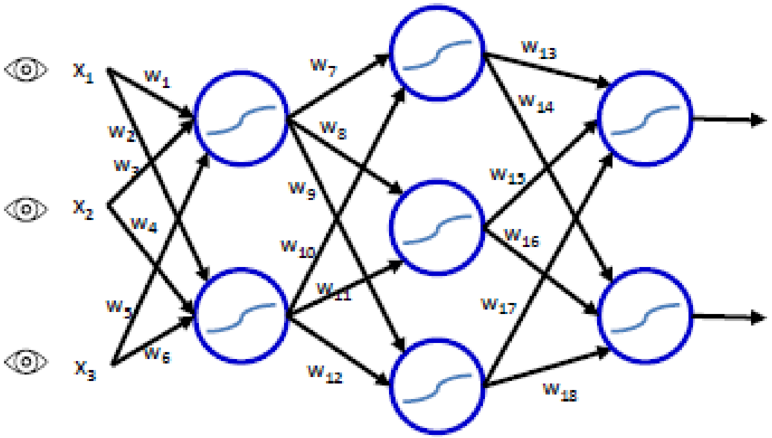
$$o = f(\sum_i w_i x_i)$$

6. Outputs are passed to other neurons (via weighted links):

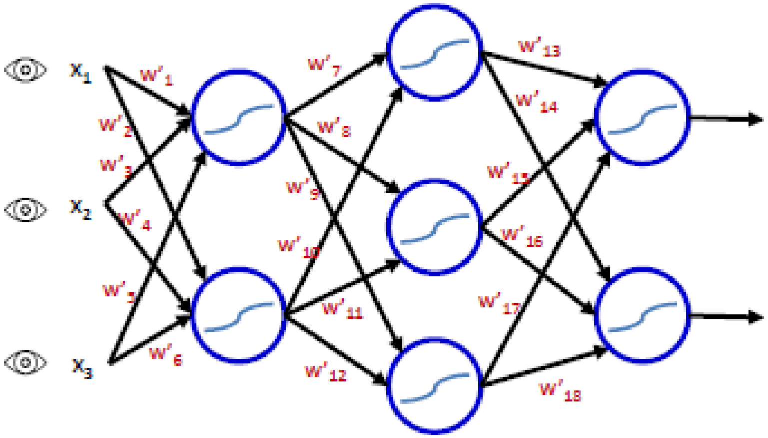


Neural Networks - MLP

7. Neurons are organized in layers:



8. Learning algorithm updates the weights:



Bias inputs are not shown

Neural Networks - MLP

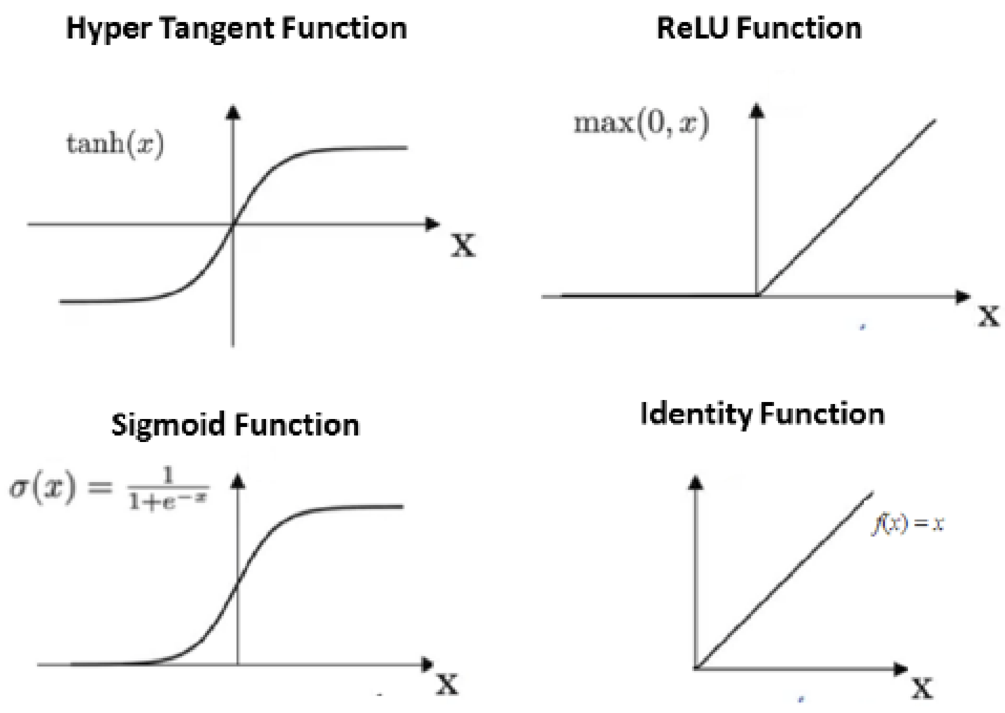
- Weights are initially unknown
  - Initialized with small values
  - Are updated by a learning algorithm.
- NN can produce a non-linear mapping of the input to the output.
  - Coding of attribute values is non-critical as long as the inputs are numeric.
  - Inputs for NN are often normalized. Why?
  - Few exceptions: i.e. SOM

Neural Networks - MLP

- Main challenges:
  - Network design:
    - How to organize the neurons?
    - How many layers, how many neurons in each layer?
    - Which activation function?
  - Learning algorithm:
    - How to update the weights?
    - How to update the weights effectively?

Neural Networks - MLP

- It has been proven:
  - Three layers are enough (if neurons are linear)
  - Two layers are enough (if neurons are non-linear).
- Common activation functions:



Neural Networks - MLP

- Weight updates
  - Compute the network error $E = \sum_i^n (o_i - t_i)^2$,where o_i is the i-th network output, and t_i the desired network output (the target).
    - When updating the weights, the aim is to minimize E *for all inputs*.
    - Many algorithms are based on gradient descent methods.
    - Update weights: $\Delta w_{ij} = -\alpha \frac{\delta E}{\delta w_{ij}}$, where $\alpha \in (0,1)$ is a learning rate.
    - Repeat for a number of epochs:
      - Select a training sample
      - Compute the output, then compute the error.
      - Compute the gradient then update the weights.

Neural Networks – MLP vs DNN

- Formally it has been proven:
  - Three layers are enough (if neurons are linear)
  - Two layers are enough (if neurons are non-linear)

- A surprise: Deep Neural Networks
  - For complex problems it was found that deep NN are much better.
    - Many layers (possibly hundreds)
    - CNN (1995)
    - Breakthrough after 2000 (massive parallel GPUs)

Neural Networks in R

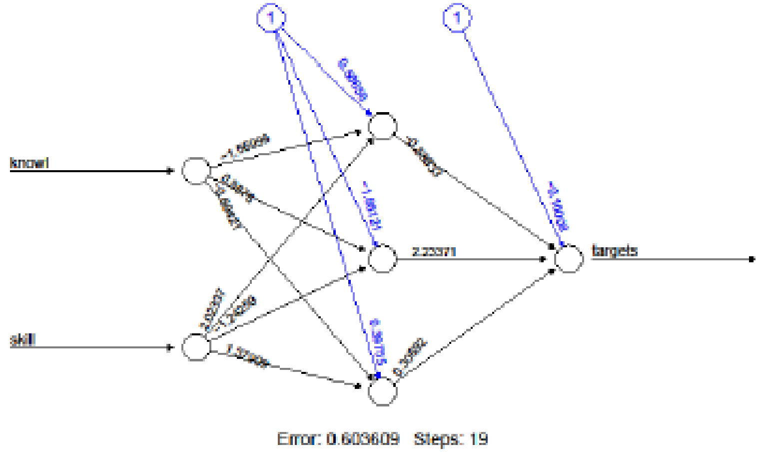
- Example: A training dataset

Knowledge Score	Communication Skills Score	Placement found
20	90	Yes
10	20	No
30	40	No
20	50	No
80	50	Yes
30	80	Yes

- Can a neural network predict placement given knowledge score and communication score of a student?

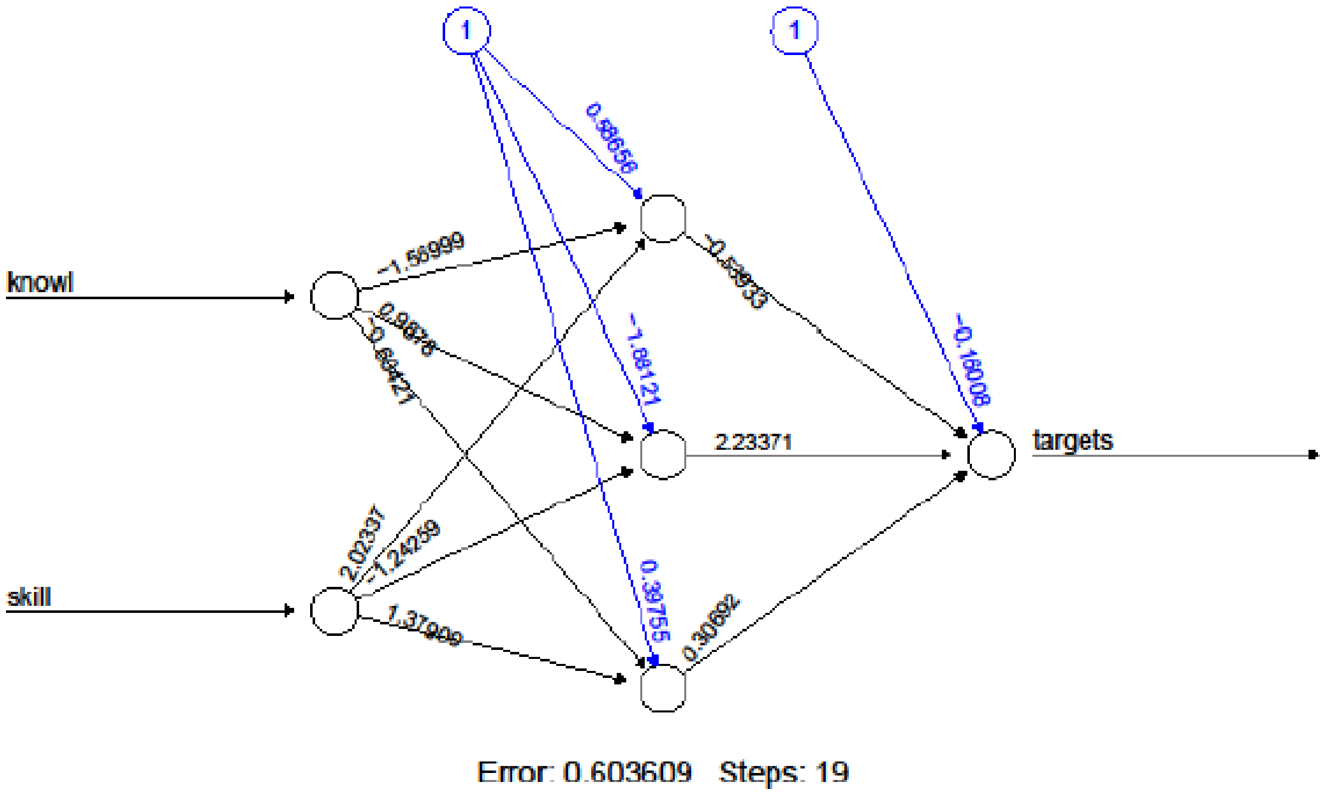
Neural Networks in R

```
#install.packages("neuralnet")
require(neuralnet)

#create training set
knowl=c(20,10,30,20,80,30)
skill=c(90,20,40,50,50,80)
targets=c(1,0,0,0,1,1)
df=data.frame(knowl,skill,targets)
```




```
#train NN
nn=neuralnet(targets~knowl+skill, data=df, hidden=3, act.fct =
"logistic", linear.output = FALSE)

#display network
plot(nn)
```

Neural Networks in R



Neural Networks in R

```
#predict with test data
knowl=c(30,40,85)
skill=c(85,50,40)
test=data.frame(knowl, skill)
Predict = compute(nn, test)
Predict$net.result

0.9928202080
0.3335543925
0.9775153014
```

- Your results may vary. Why?
- We expected results such as 0s and 1s. What to do?

Neural Networks in R

```
# Converting probabilities into binary classes setting threshold
# level 0.5
prob <- Predict$net.result
pred <- ifelse(prob>0.5, 1, 0)
pred

1
0
1
```

Reason to choose

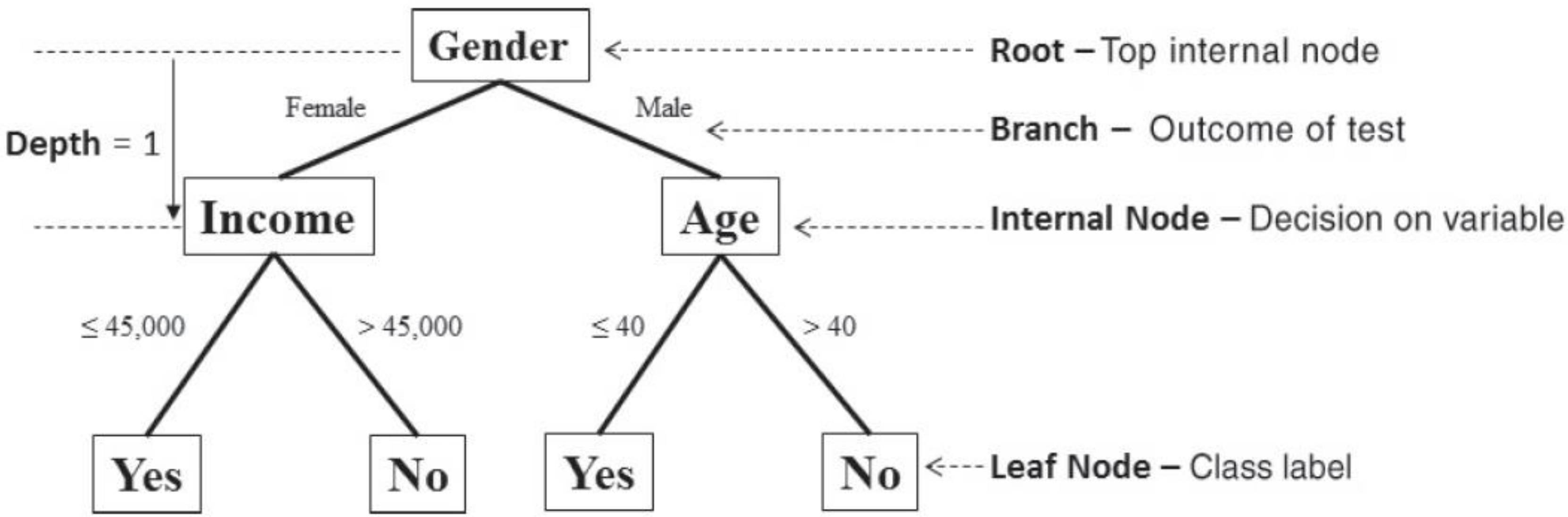
- Neural Nets are massive parallel systems
  - Can be implemented efficiently on multi-core (i.e. GPU) systems.
  - Trained models are computationally very efficient when processing new inputs.
- Neural Nets can solve a wide range of problems, and can classify samples into an arbitrary number of classes.
  - NN perform better than humans on growing number of tasks (i.e. playing chess, Go, lip reading,...)
- Limited data pre-processing required.
- Insensitive to noise
- Often a tool of choice in Big Data Analytics.

Caution

- Most supervised Neural Networks are “black box” classifiers.
  - They are unable to show or explain how a result came to be.
  - i.e. what in the input caused the network to respond in a certain way?
- They have problems with unbalanced learning problems.
  - i.e. when there are many more samples in one class than in another class.
- The model is prone to overfit the training data when choosing too many neurons and/or layers.
  - Performance may be sub-optimal if choosing to few neurons or layers.
  - Finding the best number of neurons and layers is an art.
- Training can be time consuming.
  - They tend to require a lot of training samples to perform well.

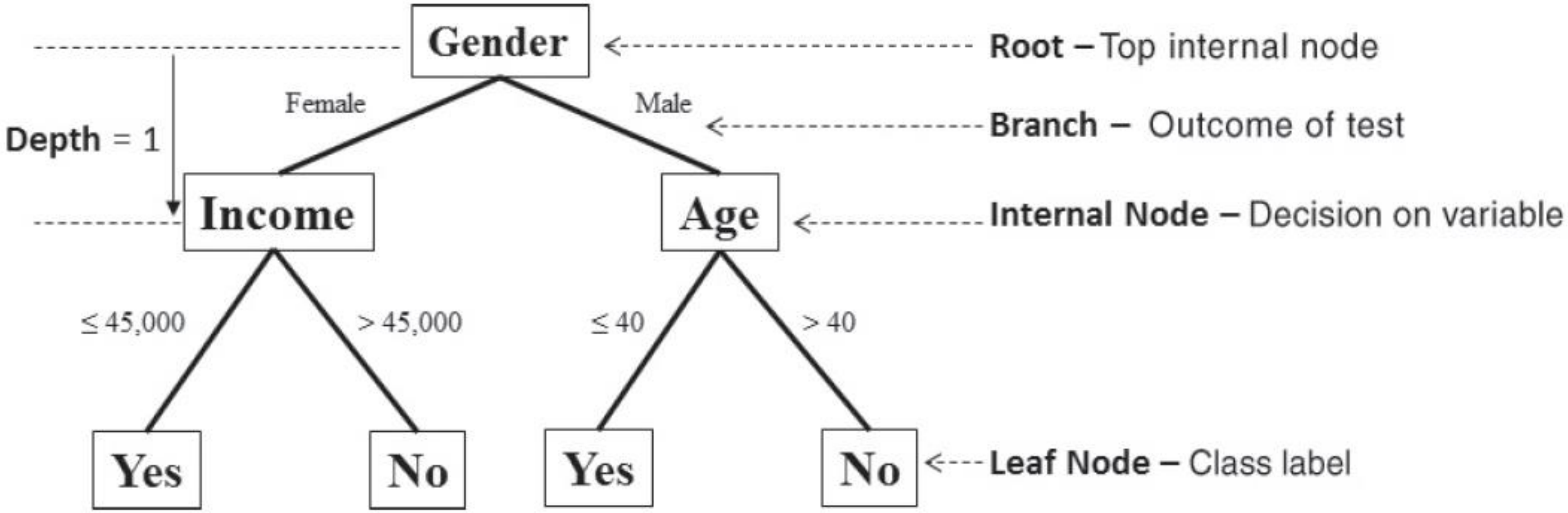
Decision Tree

- A decision tree uses a tree structure to specify sequences of decisions and consequences
- Given input variable $X = \{x_1, x_2, \ldots, x_n\}$, the goal is to predict an output variable Y



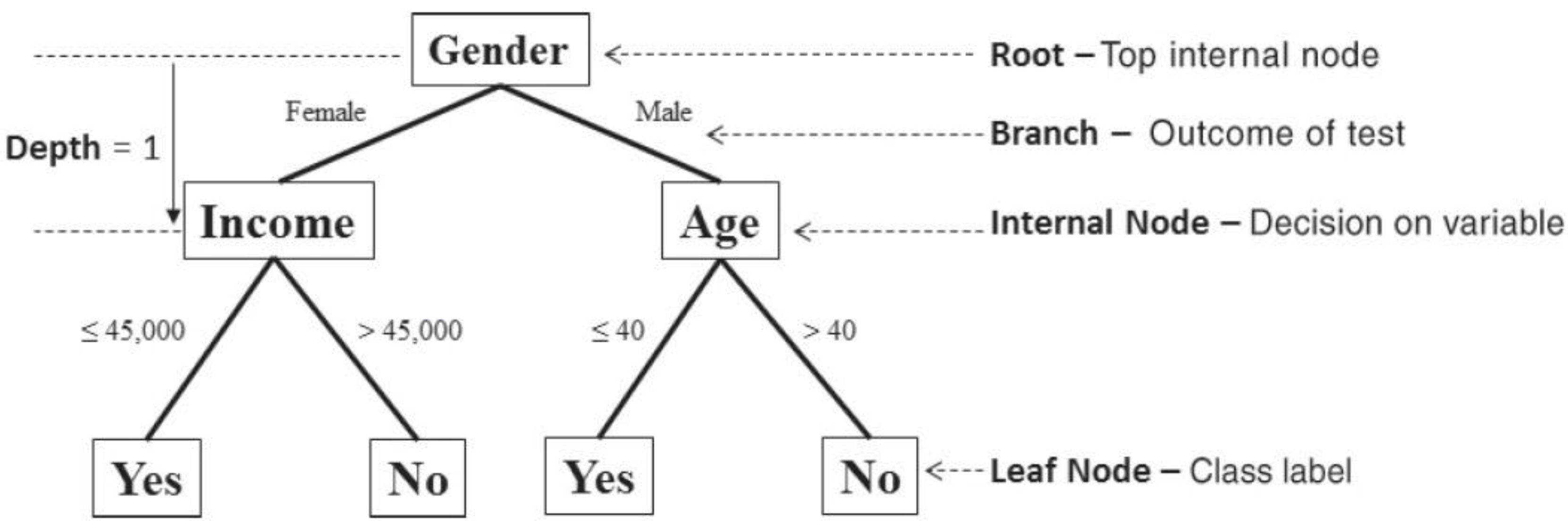
Decision Tree

- Each node tests a particular input variable
- Each branch represents the decision made
- Classifying a new observation is to traverse this decision tree.



Decision Tree

- The depth of a node is the minimum number of steps required to reach the node from root
- Leaf nodes are at the end of the last branches on the tree, representing class labels



The General Algorithm of DT

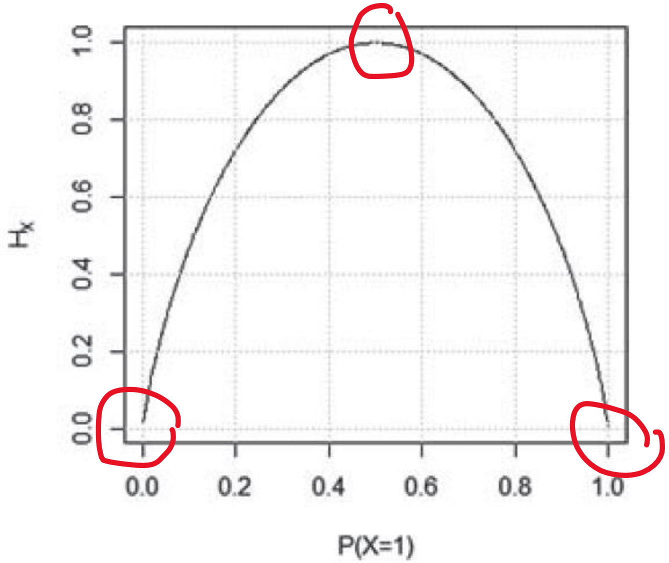
- The objective of a decision tree algorithm
  - Construct a tree T from a training set S
- The algorithm picks the most informative attribute to branch the tree and does this recursively for each of the sub-trees.
- The most informative attribute is identified by
  - Information gain, calculated based on Entropy

The General Algorithm of DT

- Entropy

Given a class X and its label $x \in X$, let $P(x)$ be the probability of x. H_x, the entropy of X, is defined as

$$H_X = -\sum_{\forall x \in X} P(x) \log_2 P(x)$$



Entropy of coin flips, where X=1 represents heads

Question:
In a bank marketing dataset, there are 2000 customers in total. Among them, 1789 subscribed term deposit. What is the entropy of the output variable "subscribed" ($H_{subscribed}$)?

The General Algorithm of DT

- Conditional entropy

Given an attribute X, its value x, its outcome Y, and its value y, conditional entropy $H_{Y|X}$ is the remaining entropy of Y given X,

$$H_{Y|X} = \sum_{x} P(x) H(Y|X = x)$$

$$= -\sum_{\forall x \in X} P(x) \sum_{\forall y \in Y} P(y|x) \log_2 P(y|x)$$

The General Algorithm of DT

- Information gain

The information gain of an attribute A is defined as the difference between the base entropy and the conditional entropy of the attribute,

$$InfoGain_A = H_S - H_{S|A}$$

- It compares
  - The degree of purity of the parent node before a split
  - The degree of purity of the child node after a split

The General Algorithm of DT

- The algorithm constructs sub-trees recursively until one of the following criteria is met
  - All the leaf nodes in the tree satisfy the minimum purity threshold (i.e., are pure enough)
  - There is no sufficient information gain by splitting on more attribute (i.e., not worth anymore)
  - Any other stopping criterion is satisfied (such as the maximum depth of the tree)

Decision Tree

- An example: A bank markets its term deposit product. So the bank needs to predict which clients would subscribe to a term deposit
  - The bank collects a dataset of 2000 previous clients with known “subscribe or not”.
  - Input variables to describe each client are
    - Job, marital status, education level, credit default, housing loan, personal loan, contact type, previous campaign contact

Decision Tree

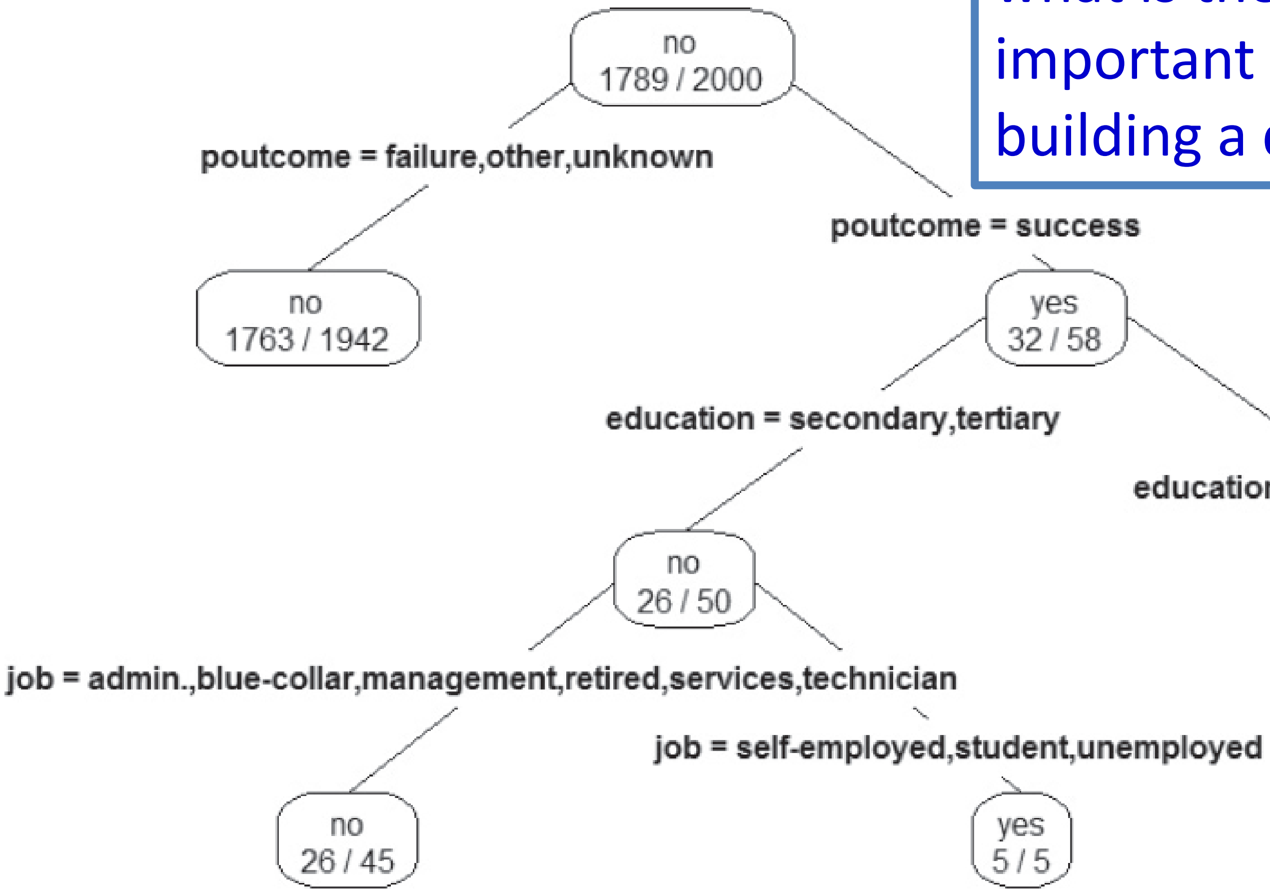
	job	marital	education	default	housing	loan	contact	poutcome	subscribed
1	management	single	tertiary	no	yes	no	cellular	unknown	no
2	entrepreneur	married	tertiary	no	yes	yes	cellular	unknown	no
3	services	divorced	secondary	no	no	no	cellular	unknown	yes
4	management	married	tertiary	no	yes	no	cellular	unknown	no
5	management	married	secondary	no	yes	no	unknown	unknown	no
6	management	single	tertiary	no	yes	no	unknown	unknown	no
7	entrepreneur	married	tertiary	no	yes	no	cellular	failure	yes
8	admin.	married	secondary	no	no	no	cellular	unknown	no
9	blue-collar	married	secondary	no	yes	no	cellular	other	no
10	management	married	tertiary	yes	no	no	cellular	unknown	no
11	blue-collar	married	secondary	no	yes	no	cellular	unknown	no
12	management	divorced	secondary	no	no	no	unknown	unknown	no
13	blue-collar	married	secondary	no	yes	no	cellular	unknown	no
14	retired	married	secondary	no	no	no	cellular	unknown	no
15	management	single	tertiary	no	yes	no	cellular	unknown	no

...

The training dataset of the bank example

Decision Tree

From your point of view, what is the most important issue in building a decision tree?



A decision tree built over the bank marketing training dataset

The General Algorithm of DT

- Assume the attribute `X` is "`contact`"
  - Its value `x` takes one value in {`cellular`, `telephone`, `unknown`}
- The outcome `Y` is "`subscribed`"
  - Its value `y` takes one value in {`no`, `yes`}

	Cellular	Telephone	Unknown
P(contact)	0.6435	0.0680	0.2885
P(subscribed=yes | contact)	0.1399	0.0809	0.0347
P(subscribed=no | contact)	0.8601	0.9192	0.9653

The General Algorithm of DT

The conditional entropy of the `contact` attribute is computed as shown here.

$$H_{Y|X} = \sum_{x} P(x) H(Y|X=x)$$
$$= -\sum_{\forall x \in x} P(x) \sum_{\forall y \in Y} P(y|x) \log_2 P(y|x)$$

$$\begin{aligned} H_{subscribed|contact} = -[&0.6435 \cdot (0.1399 \cdot \log_2 0.1399 + 0.8601 \cdot \log_2 0.8601) \\ &+ 0.0680 \cdot (0.0809 \cdot \log_2 0.0809 + 0.9192 \cdot \log_2 0.9192) \\ &+ 0.2885 \cdot (0.0347 \cdot \log_2 0.0347 + 0.9653 \cdot \log_2 0.9653)] \\ = 0.4661 \end{aligned}$$

	Cellular	Telephone	Unknown
P(contact)	0.6435	0.0680	0.2885
P(subscribed=yes | contact)	0.1399	0.0809	0.0347
P(subscribed=no | contact)	0.8601	0.9192	0.9653

The General Algorithm of DT

- The algorithm splits on the attribute with the largest information gain at each round

Attribute	Information Gain
poutcome	0.0289
contact	0.0201
housing	0.0133
job	0.0101
education	0.0034
marital	0.0018
loan	0.0010
default	0.0005

The General Algorithm of DT

- Information gain

The information gain of an attribute A is defined as the difference between the base entropy and the conditional entropy of the attribute,

$$InfoGain_A = H_S - H_{S|A}$$

$$InfoGain_{contact} = H_{subscribed} - H_{subscribed|contact}$$
$$= 0.4862 - 0.4661 = 0.0201$$

- It compares
  - The degree of purity of the parent node before a split
  - The degree of purity of the child node after a split

Properties of Decision Tree

- Computationally inexpensive, easy to classify
- Classification rules can be understood
- Handle both numerical and categorical input
- Handle variables that have a nonlinear effect on the outcome, better than linear models
- Not a good choice if there are many irrelevant input variables
  - Feature selection will be needed

Caution

- Decision tree uses greedy algorithms
  - It always chooses the option that seems the best available at that moment
  - However, the option may not be the best overall and this could cause overfitting
  - An ensemble technique can address this issue by combining multiple decision trees that use random splitting

Evaluating a Decision Tree

- Evaluate a decision tree
  - Evaluate whether the splits of the tree make sense and whether the decision rules are sound (say, with domain experts)
  - Having too many layers and obtaining nodes with few members might be signs of overfitting
  - Use standard diagnostics tools for classifiers

Naïve Bayes Classifier

- A probabilistic classification method based on Bayes' theorem
- A naïve Bayes classifier assumes that the presence or absence of a particular feature of a class is unrelated to the presence or absence of other features (conditional independence assumption)
- Output includes a class label and its corresponding probability score

Naïve Bayes Classifier on Bayes' Theorem

Thomas Bayes
1702-1761

$$P(C|A) = \frac{P(A|C) \cdot P(C)}{P(A)}$$

C is the class label $C \in \{c_1, c_2, \ldots c_n\}$

A is the observed attributes $A = \{a_1, a_2, \ldots a_m\}$

$$\text{Posteriori probability} = \frac{\text{likelihood} \cdot \text{priori probability}}{\text{evidence}}$$

Bayes' Theorem

- A more practical form of Bayes' theorem

$$P(c_i|A) = \frac{P(a_1, a_2, \ldots, a_m | c_i) \cdot P(c_i)}{P(a_1, a_2, \ldots, a_m)}, i = 1, 2, \ldots n$$

C is the class label $C \in \{c_1, c_2, \ldots c_n\}$

A is the observed attributes $A = \{a_1, a_2, \ldots a_m\}$

- Given A, how to calculate $P(c_i|A)$?

Naïve Bayes Classifier

- With two simplifications, Bayes' theorem induces a Naïve Bayes classifier
- First, Conditional independence assumption
  - Each attribute is conditionally independent of every other attribute given a class label c_i

$$P(a_1, a_2, \ldots, a_m | c_i) = P(a_1|c_i) P(a_2|c_i) \cdots P(a_m|c_i) = \prod_{j=1}^{m} P(a_j|c_i)$$

  - This simplifies the computation of $P(A|c_i)$

Naïve Bayes Classifier

- Second, ignore the denominator *P(A)*
  - Removing the denominator has no impact on the relative probability scores
- In this way, the classifier becomes

$$P(c_i|A) \propto P(c_i) \cdot \prod_{j=1}^{m} P(a_j|c_i) \qquad i = 1,2,\ldots n$$

$$P(c_i|A) \propto \log P(c_i) + \sum_{j=1}^{m} \log P(a_j|c_i) \qquad i = 1,2,\ldots n$$

Caution

- An issue on rare event
  - What if one of the attribute values does NOT appear in a class c_i in a training dataset?
  - $P(a_j|c_i)$ for this attribute value will equal zero!
  - $P(c_i|A)$ will simply become zero!
- Smoothing technique
  - It assigns a small nonzero probability to rare events not included in a training dataset

Naïve Bayes Classifier

- Laplace smoothing (add-one smoothing)
  - It pretends to see every outcome once more than it actually appears

$$P^{**}(x) = \frac{count(x) + \varepsilon}{\Sigma_x [count(x) + \varepsilon]} \qquad \varepsilon \in [0,1]$$

Naïve Bayes Classifier

- Advantages
  - Simple to implement, commonly used for text classification
  - Handle high-dimensional data efficiently
  - Robust to overfitting with smoothing technique
- Disadvantages
  - Sensitive to correlated variables (Why?)
  - Not reliable for probability estimation

Naïve Bayes Classifier

- An example
  - With the bank marketing dataset, use Naïve Bayes Classifier to predict if a client would subscribe to a term deposit
- Building a Naïve Bayes classifier requires to calculate some statistics from training dataset
  - $P(A|c_i)$ for each class i=1,2,...,n
  - $P(a_j|c_i)$ for each attribute j=1,2,...,m in each class

$$P(c_i|A) \propto P(c_i) \cdot \prod_{j=1}^{m} P(a_j|c_i) \qquad i = 1,2,\ldots n$$

Naïve Bayes Classifier

- $P(A \mid c_i)$ for each class

$$\mathrm{P}\left(subscribed = \mathrm{yes}\right) \approx 0.11 \text{ and } \mathrm{P}\left(subscribed = \mathrm{no}\right) \approx 0.89$$

The training set contains several attributes: `job`, `marital`, `education`, `default`, `housing`, `loan`, `contact`, and `poutcome`.

- $P(a_j \mid c_i)$ for each attribute in each class

$$P(single \mid subscribed = yes) \approx 0.35$$

$$P(married \mid subscribed = yes) \approx 0.53$$

$$P(divorced \mid subscribed = yes) \approx 0.12$$

$$P(single \mid subscribed = no) \approx 0.28$$

$$P(married \mid subscribed = no) \approx 0.61$$

$$P(divorced \mid subscribed = no) \approx 0.11$$

Naïve Bayes Classifier

- Testing a Naïve Bayes classifier on a new data

j	a_j	$P(a_j \mid subscribed = yes)$	$P(a_j \mid subscribed = no)$
1	job = management	0.22	0.21
2	marital = married	0.53	0.61
3	education = secondary	0.46	0.51
4	default = no	0.99	0.98
5	housing = yes	0.35	0.57
6	loan = no	0.90	0.85
7	contact = cellular	0.85	0.62
8	poutcome = success	0.15	0.01

$$P(yes|A) \propto 0.11 \cdot (0.22 \cdot 0.53 \cdot 0.46 \cdot 0.99 \cdot 0.35 \cdot 0.90 \cdot 0.85 \cdot 0.15) \approx 0.00023$$

$$P(no|A) \propto 0.89 \cdot (0.21 \cdot 0.61 \cdot 0.51 \cdot 0.98 \cdot 0.57 \cdot 0.85 \cdot 0.62 \cdot 0.01) \approx 0.00017$$

Naïve Bayes in R

- Two methods
  - Build the classifier from the scratch
  - Call `naiveBayes` function from `e1071` package

```
install.packages("e1071")    # install package e1071
library(e1071)    # load the library

# read the data into a table from the file
sample <- read.table("sample1.csv",header=TRUE,sep=",")
# define the data frames for the NB classifier
traindata <- as.data.frame(sample[1:14,])
testdata <- as.data.frame(sample[15,])
```

Naïve Bayes in R

```
model <- naiveBayes(Enrolls ~ Age+Income+JobSatisfaction+Desire,
                    traindata)
# display model
model

# predict with testdata
results <- predict (model,testdata)
# display results
results
[1] Yes
Levels:  No Yes
```

```
# use the NB classifier with Laplace smoothing
model1 = naiveBayes(Enrolls ~., traindata, laplace=.01)
```

Content

- Brief Recap
  - Clustering Analysis
    - K-means, DBSCAN, SOM
- Classification
  - Overview
  - K-Nearest Neighbor (KNN)
  - Multi-Layers Perceptron (MLP)
  - Decision Tree (DT)
  - Naïve Bayesian Classifier
- Diagnostics and Performance Indicators

Diagnostics

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 80%) & Test set (e.g., 20%)
  - Random sampling: a variation of holdout
    - Repeat holdout *k* times, avg. + std accuracy
- **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

Performance Indictors

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

Performance Indicators

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified
  Accuracy = (TP + TN)/All
- **Error rate:** *1 – accuracy*, or
  Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Performance Indicators

- **Precision**: exactness – what % that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of the positives did the classifier label as positive? (equals to sensitivity)

$$recall = \frac{TP}{TP + FN}$$

  - Perfect score is 1.0
  - In practice, inverse relationship between precision & recall

- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

Performance Indicators

- *Precision* = 90/230 = 39.13%
- *Recall* = 90/300 = 30.00% = Sensitivity

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	210	**300**	30.00 (*sensitivity*
cancer = no	140	**9560**	**9700**	98.56 (*specificity)*
Total	230	9770	**10000**	96.50 (*accuracy*)

Summary

- Supervised methods:
  - Model observed data to predict future outcomes.
- Care must be taken in performing and interpreting classification results
  - How determine the best input variables and their relationship to outcome variables.
  - Understand and validate underlying assumptions.
  - Transform variables when necessary.
  - If in doubt, use a non-linear classification method
    - Examples: Neural Nets, Naïve Bayes, SVM, …

Additional Classification Models

- Support Vector Machines
  - Max-margin linear classifier, kernel trick.
- Supervised Neural Networks
  - RNNs, Convolutional Networks, GNNs, …
- Bagging
  - Bootstrap technique, ensemble method.
  - N x weak learners -> vote on results (i.e. random forrest)
- Boosting
  - Weighted combination, ensemble method.
  - N x weak learners in series, each tasked to improve on the previous.

Q&A

Images Courtesy of Google Image