PROBLEM SOLVING USING PATTERN RECOGNITION DAY 2

Dr Zhu Fangming

Institute of Systems Science

National University of Singapore

fangming@nus.edu.sg

DAY 2 AGENDA

2.1 Solving Pattern Recognition Problems Using Supervised Learning Techniques (II):

Decision Trees

Neural networks

Support Vector Machines

2.2 Pattern Recognition Workshop 2

2.1
Solving Pattern Recognition Problems Using Supervised Learning Techniques (II)

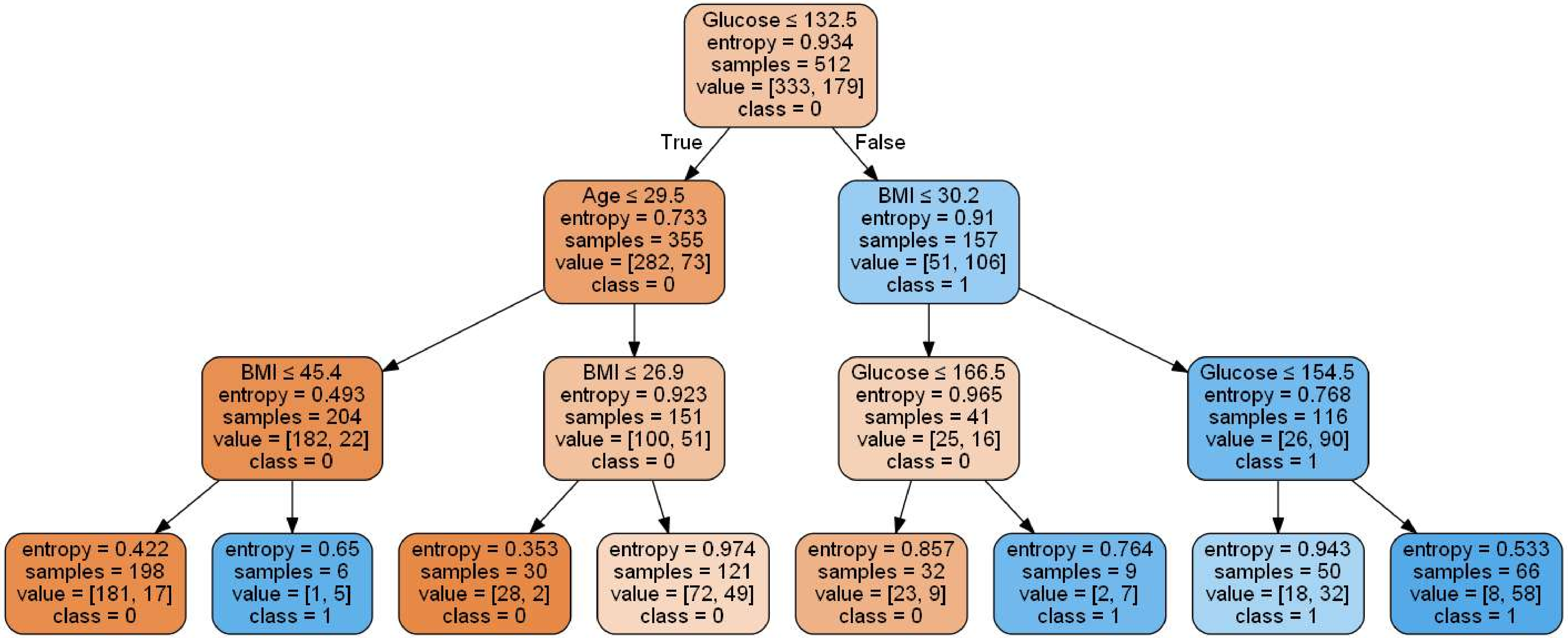
Supervised Learning Techniques (II)

- Decision Trees (DT)
- Neural Networks (NN)
- Support Vector Machines (SVM)

Decision Tree

- A decision tree is a flow-chart-like tree structure.
  - An internal node performs a test on an attribute
  - A branch represents a result of the test
  - A leaf node represents a class label
  - At each node, one feature is chosen to split training examples into distinct classes
  - A new sample is classified by following a matching path to a leaf node

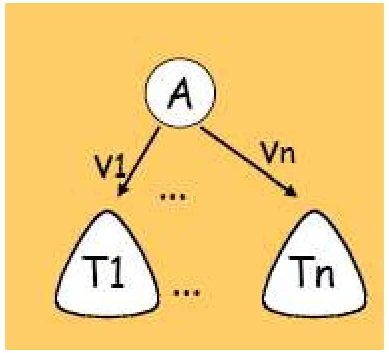
Decision Tree



Applications of Decision Trees

- Customer Relationship Management
- Fraud Detection
- Churn Prediction
- Credit Risk Prediction
- Purchasing Behavior Prediction
- Fault Detection
- Sentiment Analysis
- Investment Solutions

Basic Algorithm: Quinlan's ID3/C4.5/C5.0



- create a root node for the tree
- if all examples from S belong to the same class Cj
- then label the root with Cj
- else
    - select the "most informative" attribute A with values v1, v2, , vn
    - divide the training set S into S1, …, Sn according to values v1,…,vn
    - recursively build subtrees T1,…,Tn for S1,…,Sn
    - generate decision tree T

Building Decision Tree

- Top-down tree construction
  - At start, all training data are at the root.
  - Partition the examples recursively by choosing one feature each time.
- At each node, available attributes are evaluated on the basis of separating the classes of the training examples. A goodness function is used for this purpose.
- Typical goodness measures:
  - Information gain (ID3/C4.5)
  - Information gain ratio (C4.5)
  - Gini index (CART)

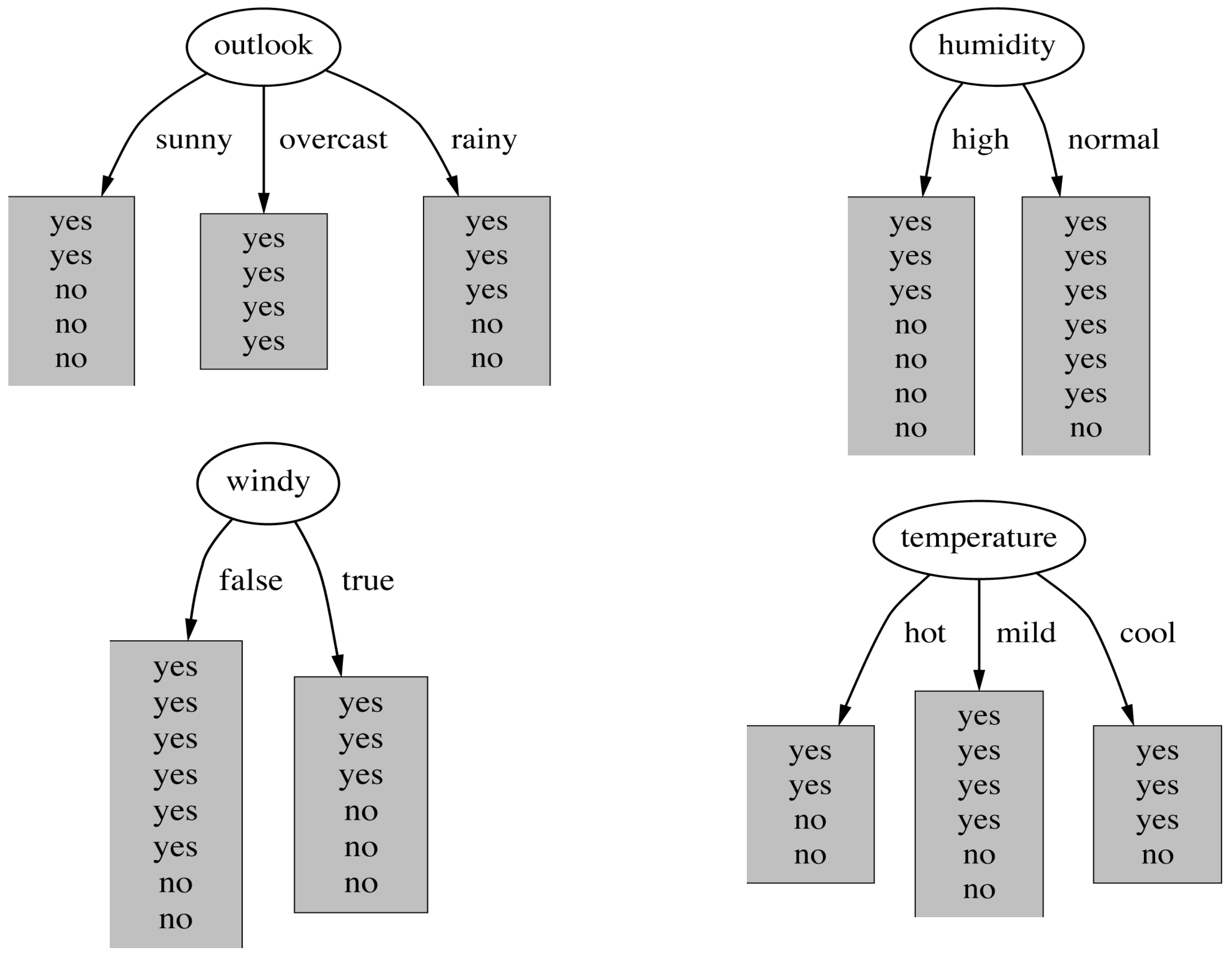
Heuristic Search

- Search bias: Search the space of decision trees from simplest to increasingly complex (greedy search, no backtracking, prefer small trees)
- Search heuristics: At a node, select the attribute that is most useful for classifying examples, split the node accordingly

Weather Data: Play Tennis or Not?

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rainy	mild	high	false	Yes
rainy	cool	normal	false	Yes
rainy	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rainy	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rainy	mild	high	true	No

Which Attribute to Select as the Root Node?



Criteria for Selecting an Attribute

- Which is the best attribute
  - The one which yields the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Popular impurity criterion: *information gain*
  - Information gain increases with the average purity of the subsets that an attribute produces
- One method: choose attribute that gives the greatest information gain

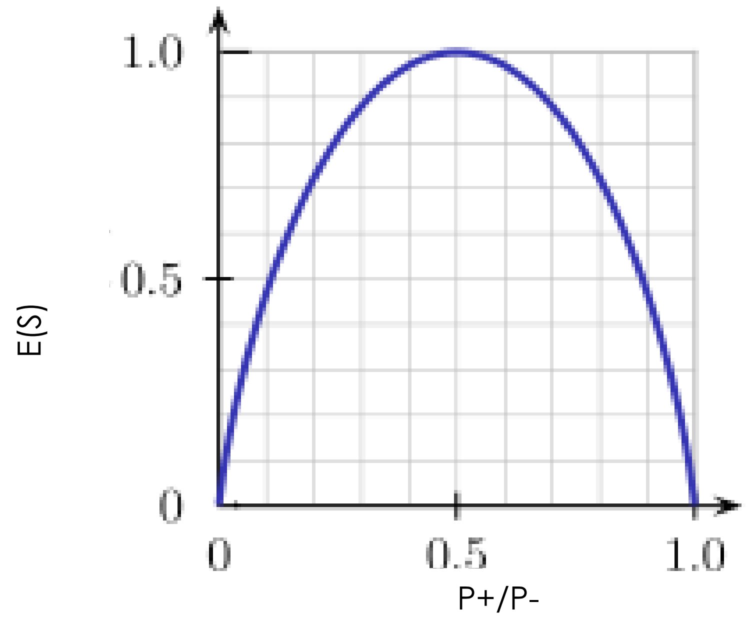
Entropy

- **S** - training set, **$C_1,\ldots,C_N$** - classes
- Formula for computing the entropy:

$$E(S) = -\sum_{c=1}^{N} p_c \cdot \log_2 p_c$$

- Interpretation:
  - Higher Entropy → Higher Uncertainty
  - Lower Entropy → Lower Uncertainty
- Entropy in binary classification problems

$$E(S) = - p_+ \log_2 p_+ - p_- \log_2 p_-$$



Information Gain

- **Information gain** measure is aimed to minimize the number of tests needed for the classification of a new object
- **Gain(S,A) -** expected reduction in entropy of **S** due to sorting on **A**

$$Gain(S,A) = E(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot E(S_v)$$

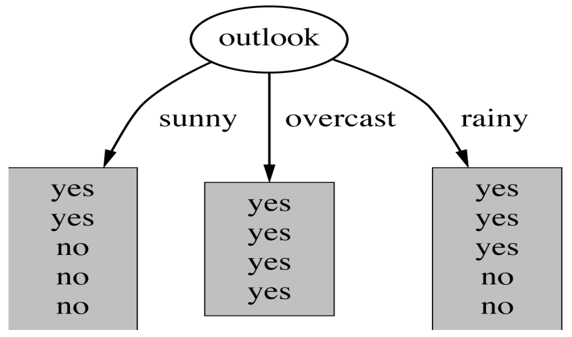
Information Gain by splitting on A

Expected Information (entropy) needed to classify S

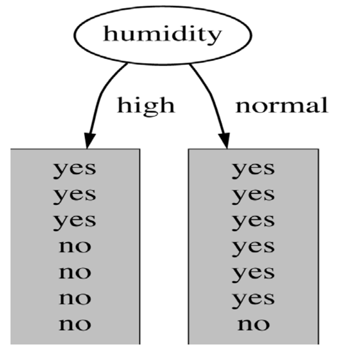
Information needed (after using A to split S into v partitions) to classify S

- **Most informative** attribute: **max Gain(S,A)**

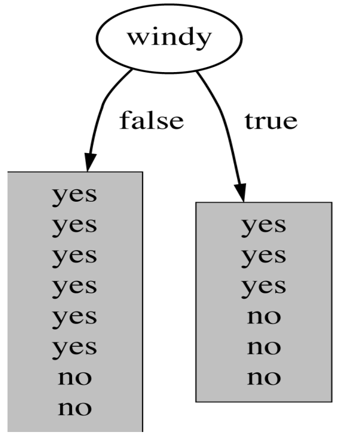
Which Attribute to Select?



$gain(\text{"Outlook"}) = 0.247$

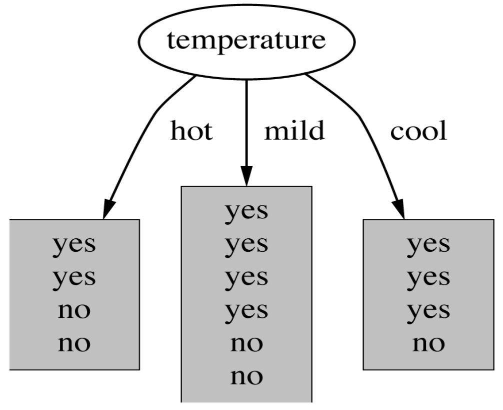


$gain(\text{"Humidity"}) = 0.152$



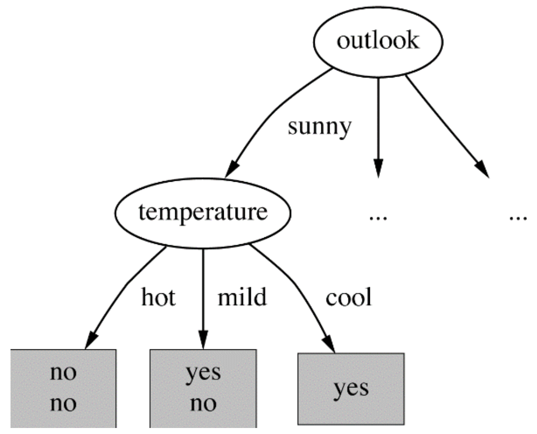
$gain(\text{"Windy"}) = 0.048$

MAX !

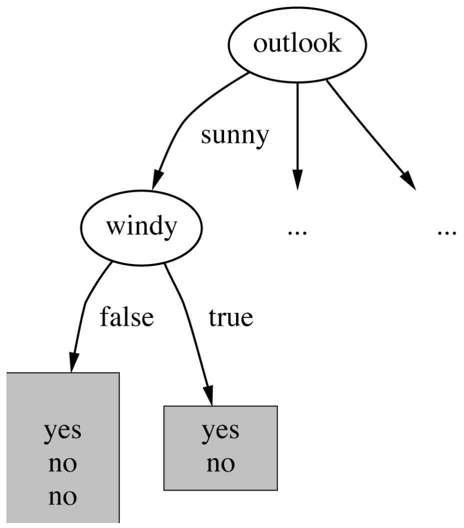


$gain(\text{"Temperature"}) = 0.029$

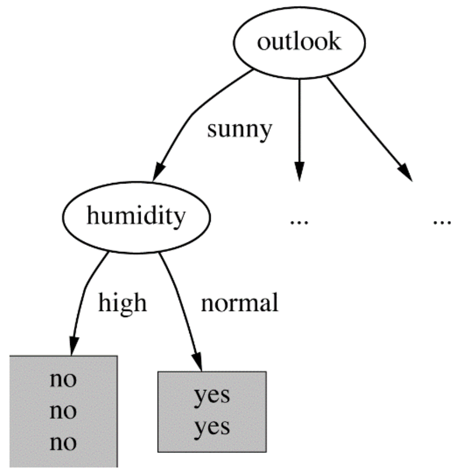
Continuing to Split



$$\text{gain}(\text{"Temperature"}) = 0.570$$



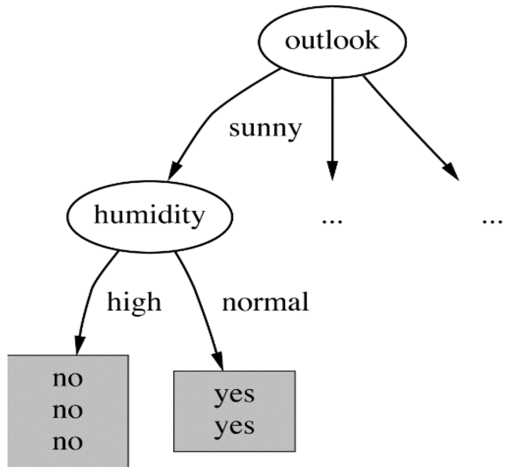
$$\text{gain}(\text{"Windy"}) = 0.019$$



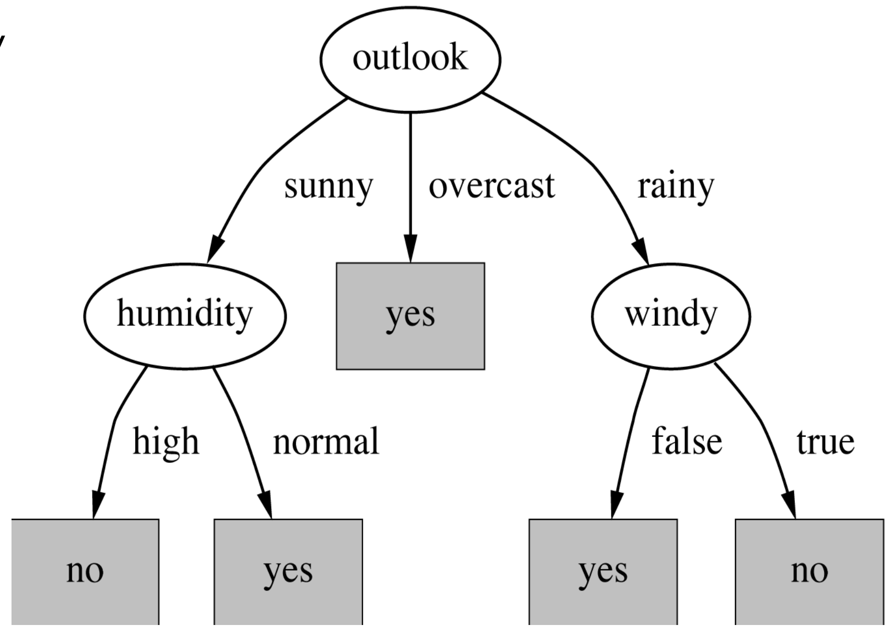
$$\text{gain}(\text{"Humidity"}) = 0.970$$

MAX !

Continuing to Split



Finally



Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point achieving the *maximum information gain* for A is selected as the split-point for A
- Split:
  - S1 is the set of tuples in S satisfying A ≤ split-point, and S2 is the set of tuples in S satisfying A > split-point

Stopping Criteria

- if all examples belong to same class C_j, label the leaf with C_j
- if all attributes were used, label the leaf with the most common value C_k of examples in the node
- **min_samples_split** - The minimum number of samples required to split an internal node.
- **min_samples_leaf** - The minimum number of samples required to be at a leaf node
- **max_depth -** The maximum depth of the tree.
- …

Highly-Branching Features

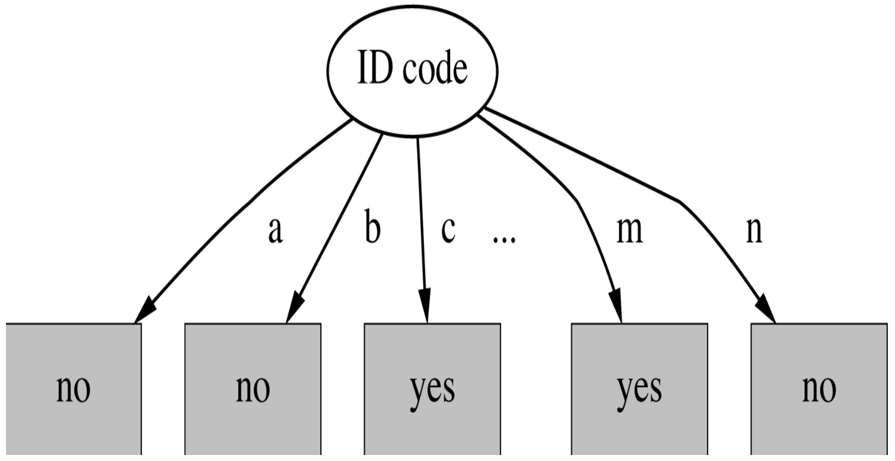
- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values

  $\Rightarrow$Information gain is biased towards choosing features with a large number of values

  $\Rightarrow$This may result in *overfitting* (selection of a feature that is non-optimal for prediction)

Split for ID Code Attribute

ID	Outlook	Temperature	Humidity	Windy	Play?
A	sunny	hot	high	false	No
B	sunny	hot	high	true	No
C	overcast	hot	high	false	Yes
D	rain	mild	high	false	Yes
E	rain	cool	normal	false	Yes
F	rain	cool	normal	true	No
G	overcast	cool	normal	true	Yes
H	sunny	mild	high	false	No
I	sunny	cool	normal	false	Yes
J	rain	mild	normal	false	Yes
K	sunny	mild	normal	true	Yes
L	overcast	mild	high	true	Yes
M	overcast	hot	normal	false	Yes
N	rain	mild	high	true	No



Entropy of split = 0 (since each leaf node is "pure", having only one case.)

Information gain is maximal for ID code

Gain Ratio

- *Gain ratio*: a modification of the information gain that reduces its bias on highly-branching attributes
- Gain ratio takes number and size of branches into account when choosing an attribute

- Intrinsic information: entropy of distribution of instances into branches

$$GainRatio(S,A) = \frac{Gain(S,A)}{IntrinsicInfo(S,A)}.$$

- *Gain ratio* (Quinlan'86) normalizes info gain by:

$$IntrinsicInfo(S,A) = -\sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

- The attribute with the maximum gain ratio is selected as the splitting attribute

Gini Index: Splitting Criteria in CART

- CART (Classification And Regression Trees)
- If a data set T contains examples from n classes, gini index -- gini(T) is defined as

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in T

gini(T) is minimized if the classes in T are skewed

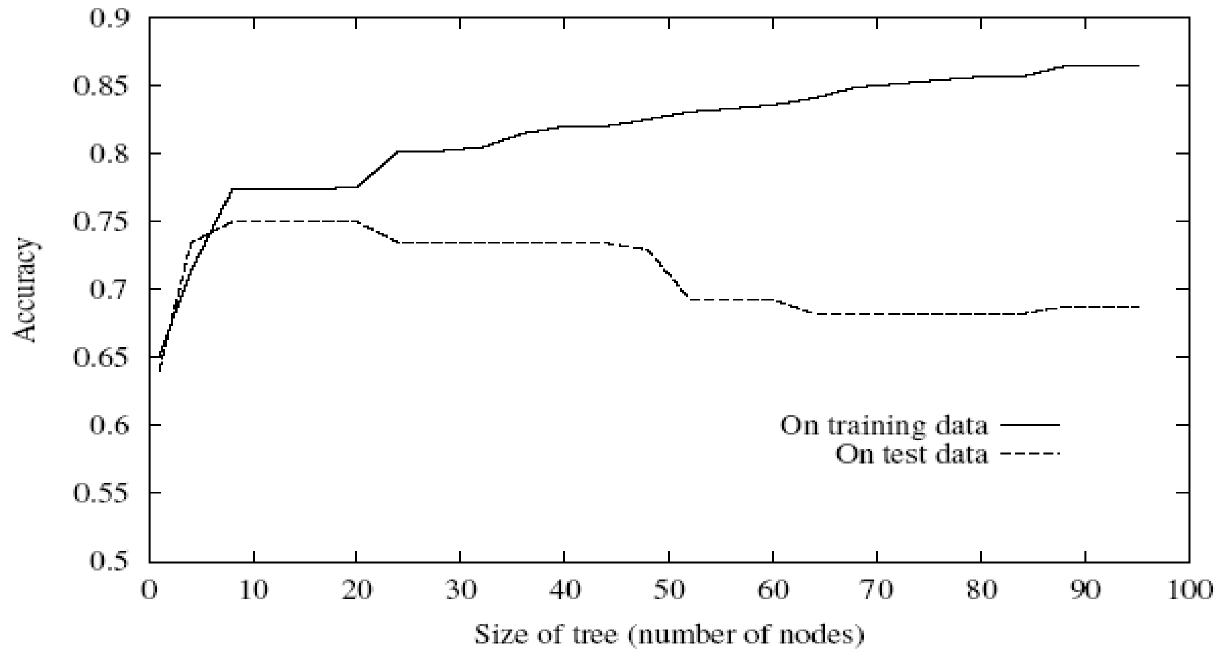
- After splitting T into two subsets T1 and T2 with sizes N1 and N2, the gini index of the split data is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute providing smallest $gini_{split}$(T) is chosen to split the node

Overfitting / Overtraining

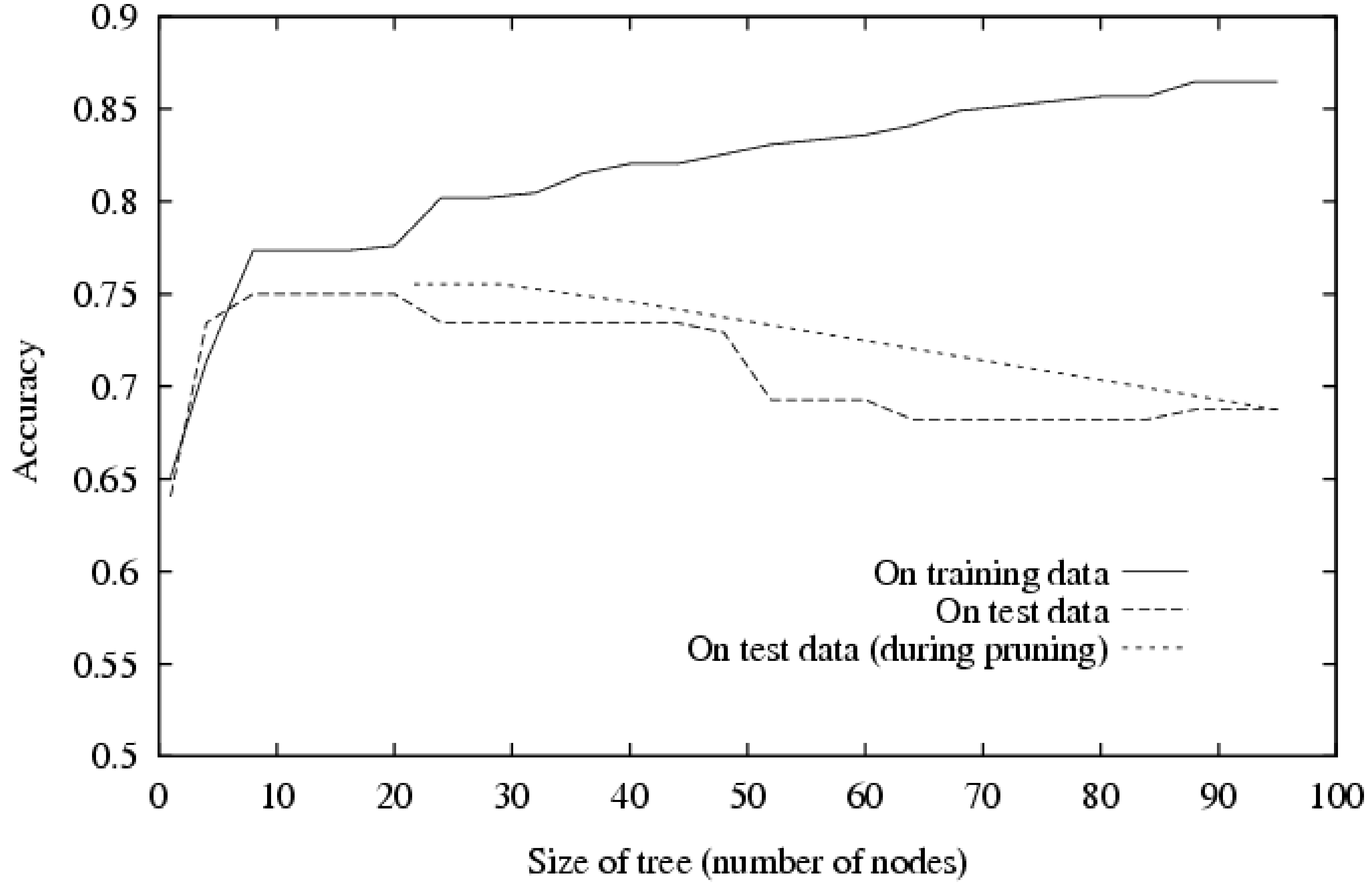
- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples



Overfitting and Tree Pruning

- Two approaches to avoid overfitting
  - Pre-pruning (forward pruning): stop growing the tree e.g.
    - When data split not statistically significant
    - Too few examples are in a split
  - Postpruning: *Remove branches* from a “fully grown” tree—get a sequence of progressively pruned trees
    - Use a set of validation data to decide which is the “best pruned tree”

Overfitting and Tree Pruning



Decision Tree Modeling using Scikit-learn

```
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier(criterion='entropy',max_depth=3, random_state=0)
clf.fit(X_train, y_train)
clf.predict(X_test)
```

Decision Tree Summary

- Decision Trees
  - splits – binary, multi-way
  - splitting criteria – info gain, gain ratio, gini, …
  - pruning
  - rule extraction from trees
- Avoid Overfitting
  - Pruning
  - Fixed depth/ Early stopping…

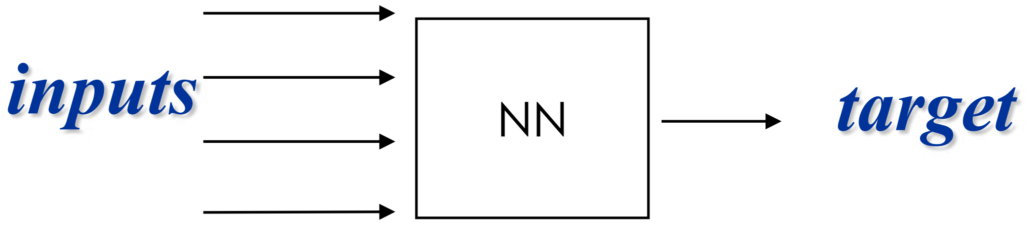
Pros and Cons of Decision Trees

- Pros:
  - simple to understand and interpret
  - little data preparation and little computation
  - indicates which attribute are most important for classification

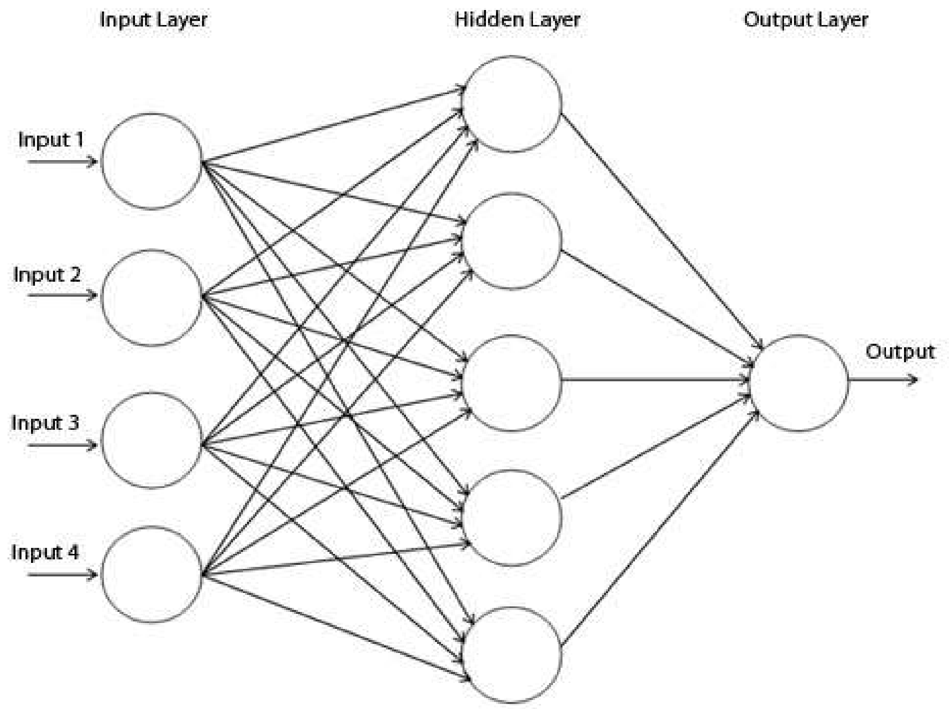
- Cons:
  - not guaranteed to produce an optimal decision tree
  - perform poorly with many classes and small data
  - over-complex trees do not generalise well from the training data (overfitting)

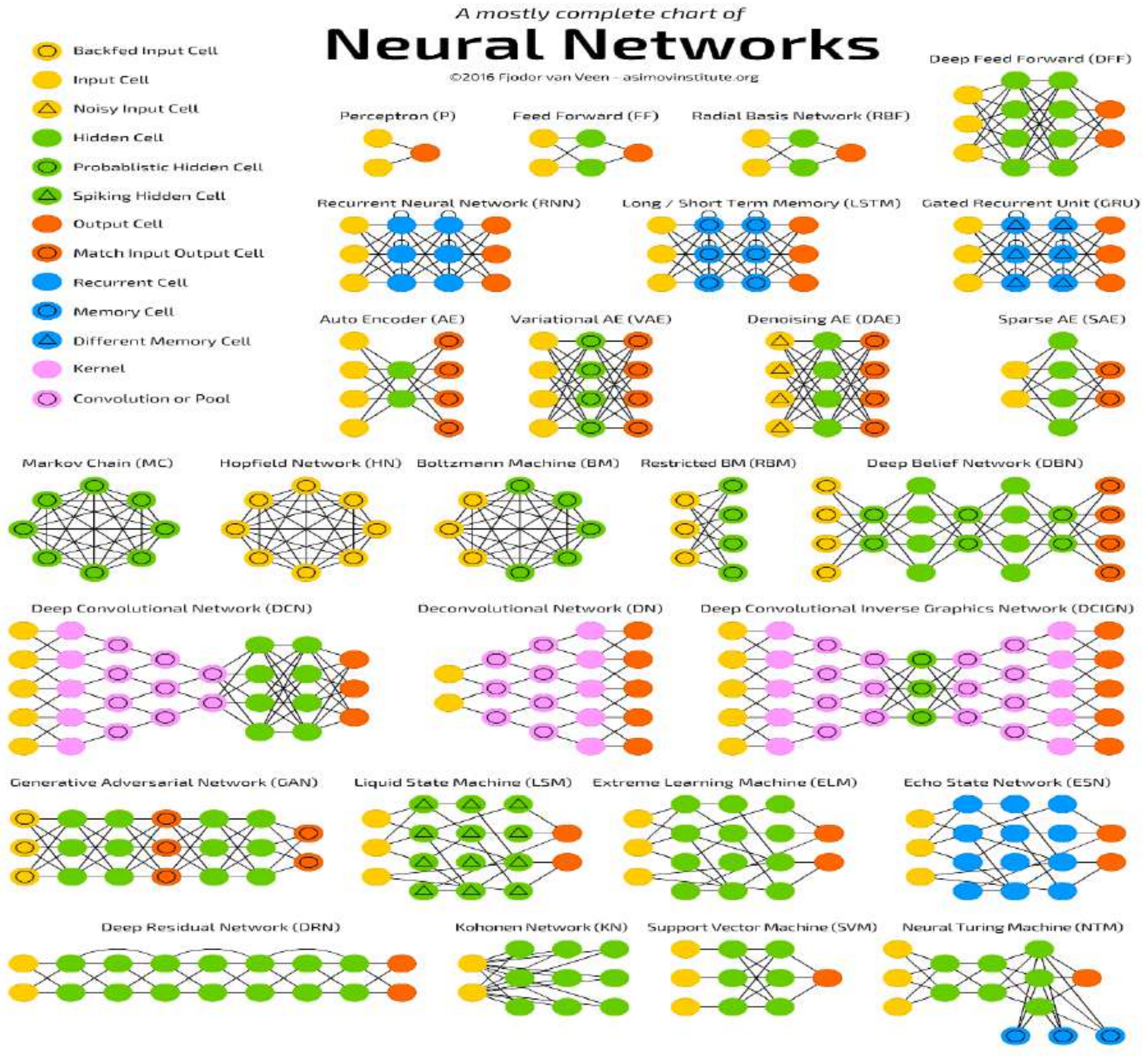
Neural Networks

- Neural Networks (NN) are biologically inspired and attempt to build computational models that operate like a human brain.
- These networks can “learn” from the data and recognize patterns.
- Make no assumptions about the data
- Can be very accurate
- Handle both numeric targets and categorical targets
- A black box….



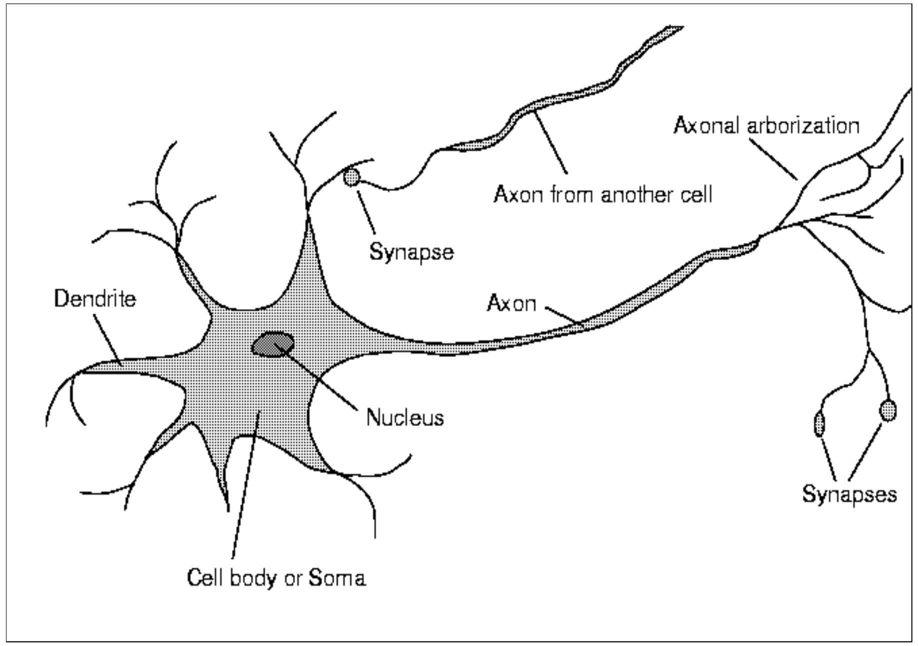
Neural Networks

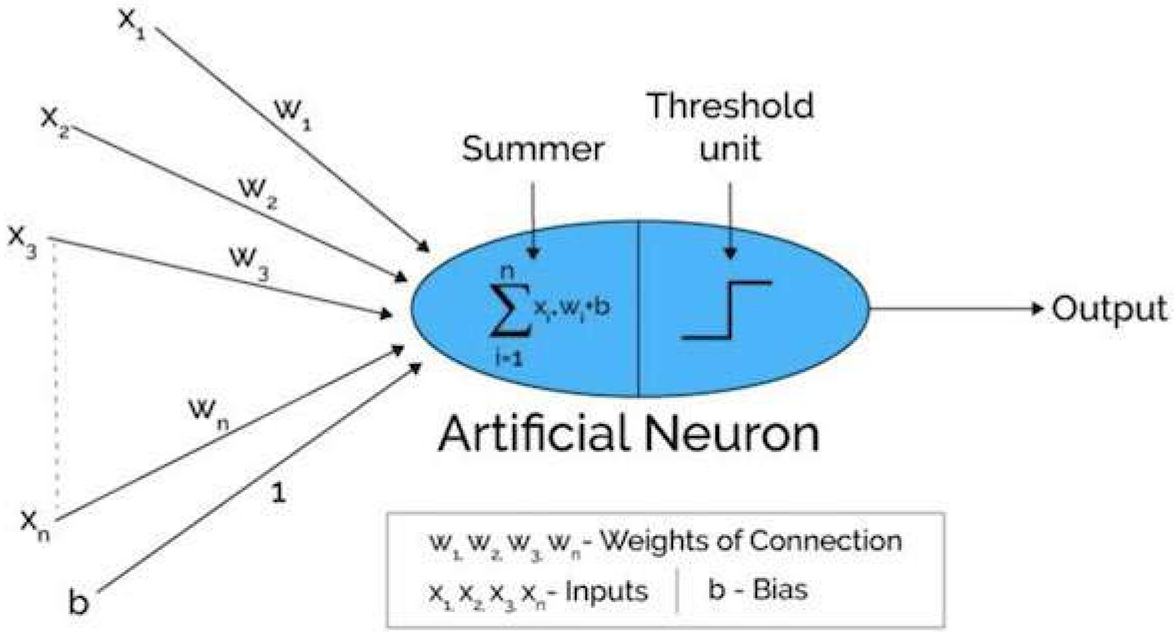



A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

http://www.asimovinstitute.org/neural-network-zoo/

From Biological Neuron to Artificial Neuron

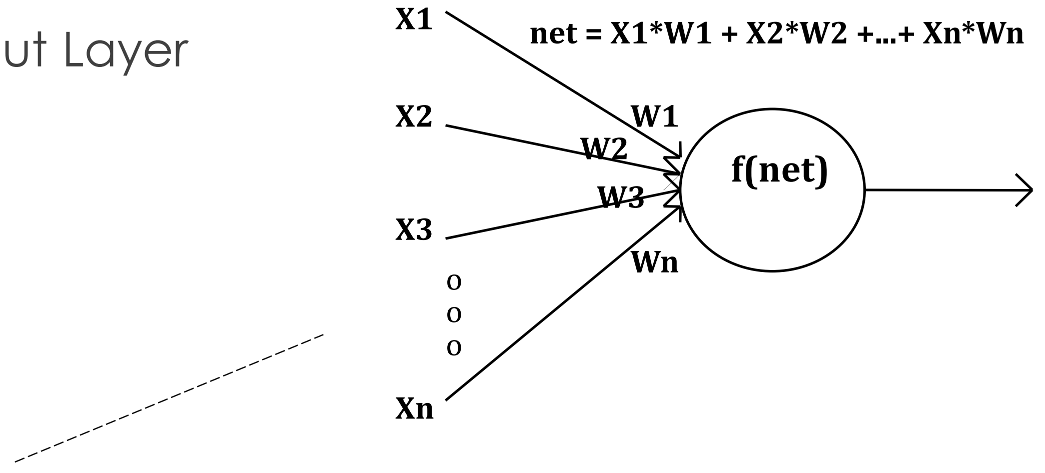


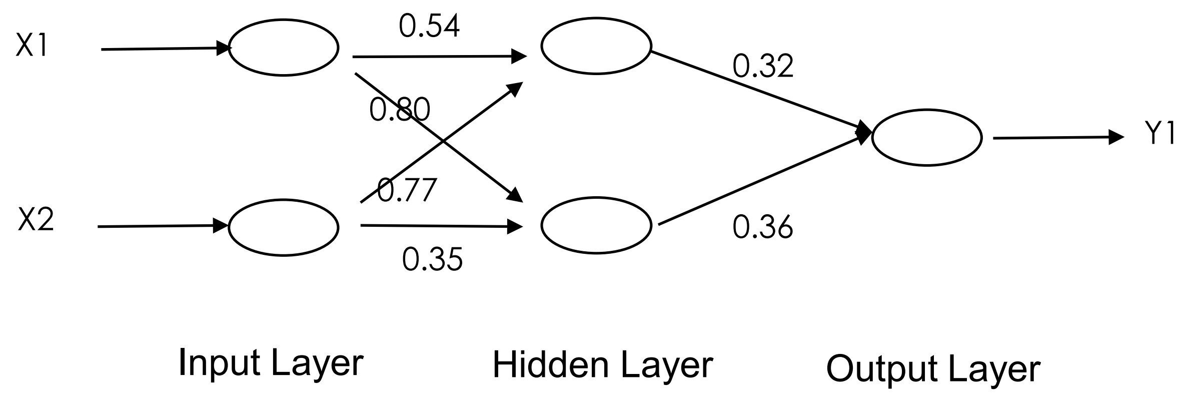


x_1
x_2
x_3
x_n
b
w_1
w_2
w_3
w_n
1
Summer
Threshold unit
$\sum_{i=1}^{n} x_i w_i + b$
Output
Artificial Neuron
$w_1\ w_2\ w_3\ w_n$ - Weights of Connection
$x_1\ x_2\ x_3\ x_n$ - Inputs
b - Bias

General Architecture of Neural Networks

- Framework (in general, but not for all NNs)
  - Input layer + Hidden Layer + Output Layer
  - Weights
  - Activation functions f(net)

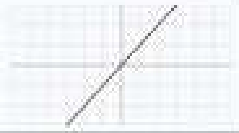
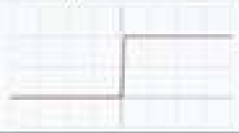
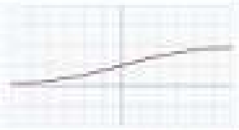
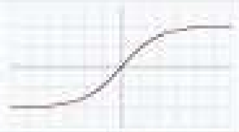
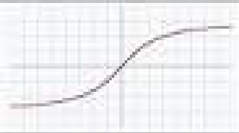
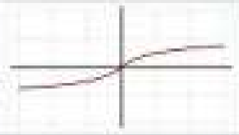




General Architecture of Neural Networks (cont.)

- Weights
  - Normally initial weights are randomised to small real numbers
- Learning rule
  - determine how to adapt connection weights in order to optimise the network performance $W_i(t+1)=W_i(t)+\Delta W_i(t)$
  - indicate how to calculate the weight adjustment during each training cycle
- Activation calculation & Weight adjustment
  - Compute the activation levels across the network
  - Weight adjustment based on the errors /distance

Activation functions

Name	Plot	Equation	Derivative (with respect to *x*)
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Softsign [7][8]		$f(x) = \frac{x}{1 + \|x\|}$	$f'(x) = \frac{1}{(1 + \|x\|)^2}$

https://www.codeproject.com/Articles/1200392/Neural-Network

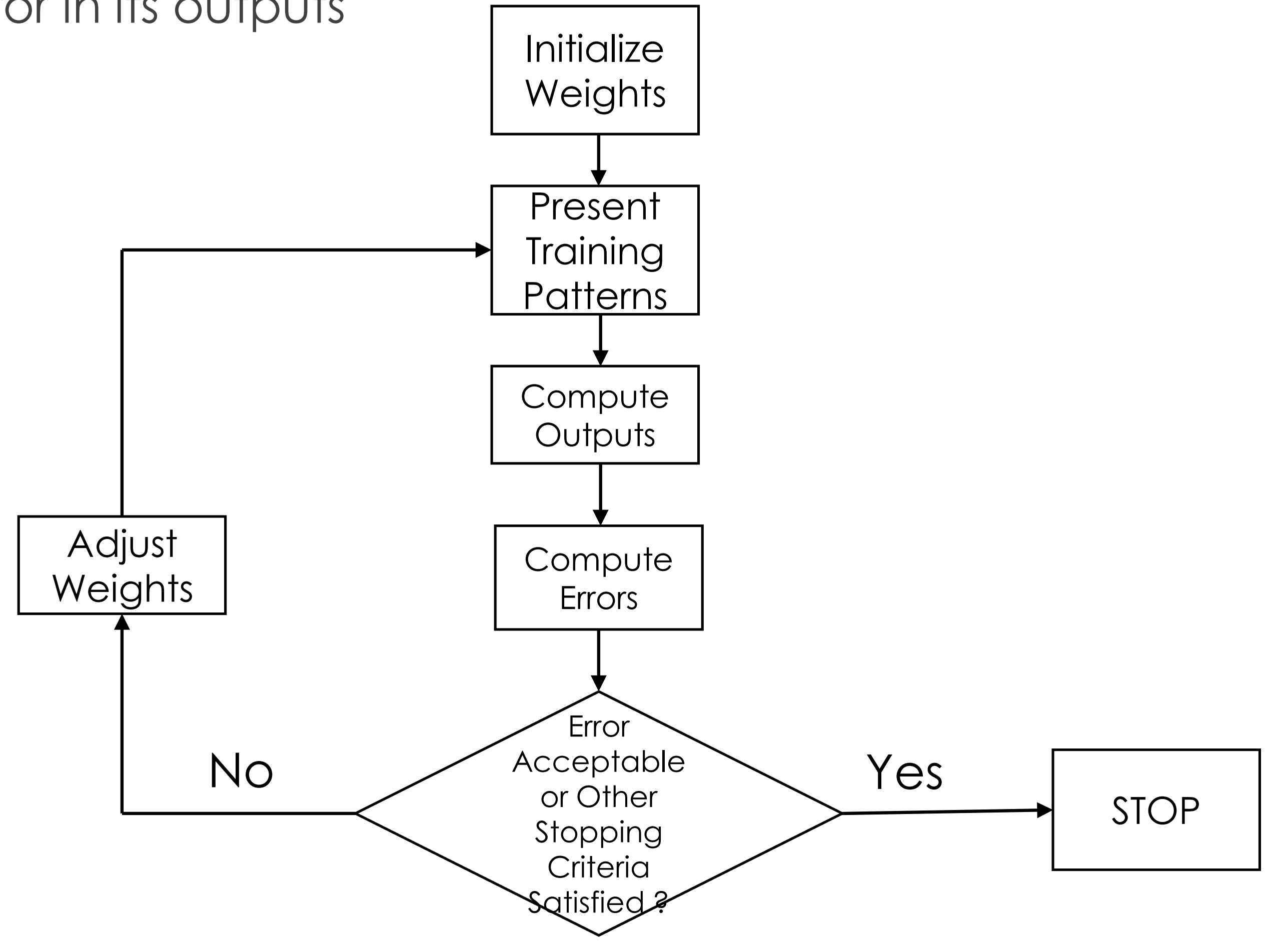
Training Neural Networks

- Require lots of training data
- Training can be slow!
- Limit training by
  - the time taken
  - number or training iterations
  - the accuracy

Neural Network Learning /Training

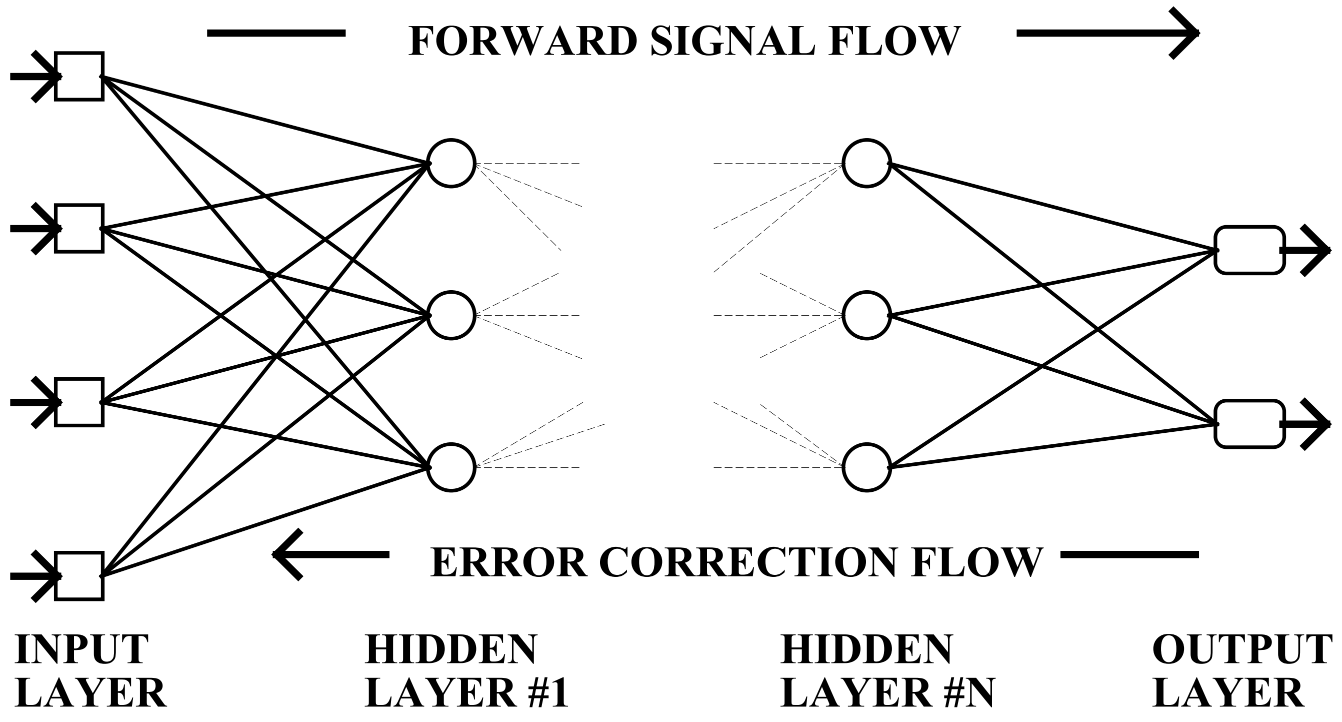
- **Supervised learning**

  Training samples are shown to the network as input and the weights are adjusted to minimize the error in its outputs


Initialize Weights
Present Training Patterns
Compute Outputs
Compute Errors
Error Acceptable or Other Stopping Criteria Satisfied ?
No
Adjust Weights
Yes
STOP

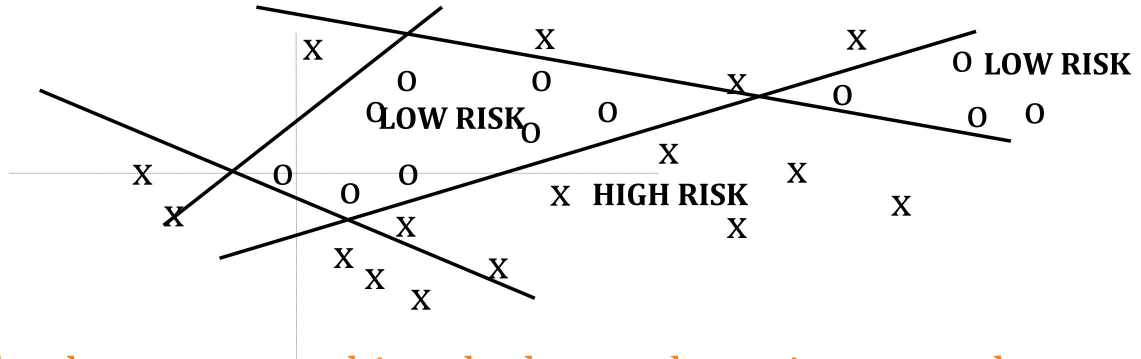
Multilayer Perceptron (MLP) with Backpropagation Learning

- Propagate signals forward and then errors backward
- Backpropogation (BP) ~ gradient descent learning
- Weights in hidden layers are adjusted to reduce aggregate errors in the output layer

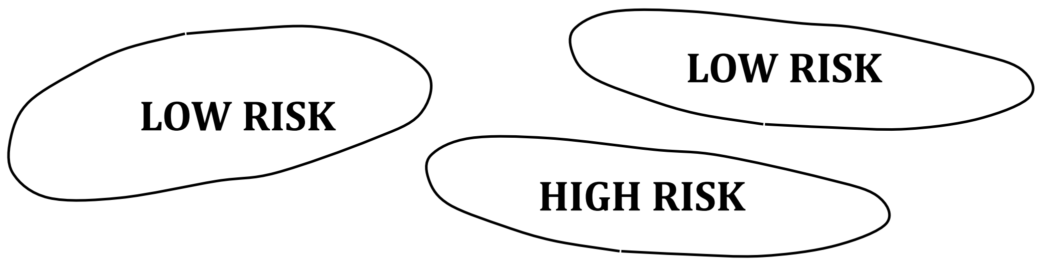


MLP Networks

- Nodes in the first hidden layer represent hyperplanes



- Nodes in second hidden layer can combine the hyperplanes into complex non-linear surfaces



- Beware…. too many nodes or layers can be very hard to train (requiring many samples & long training time)

Steps of Backpropagation Algorithm

1. Initialize the weights to small random numbers

2. Randomly select a training pattern pair (x^p, t^p) and present the input pattern x^p to the network. Compute the corresponding network output pattern z^p

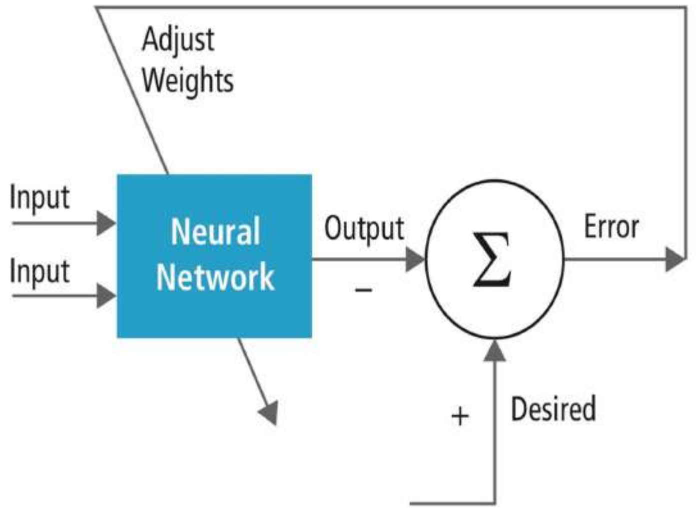
3. Compute the error E^p for pattern (x^p, t^p)

4. Backpropagate the errors according to the BP weight adjustment formulas

5. Test the Loss Function (mean square error (MSE), cross-entropy, etc):

   If it is below the required threshold, stop. Otherwise, repeat steps 2-5.

6. Test for generalization performance if appropriate

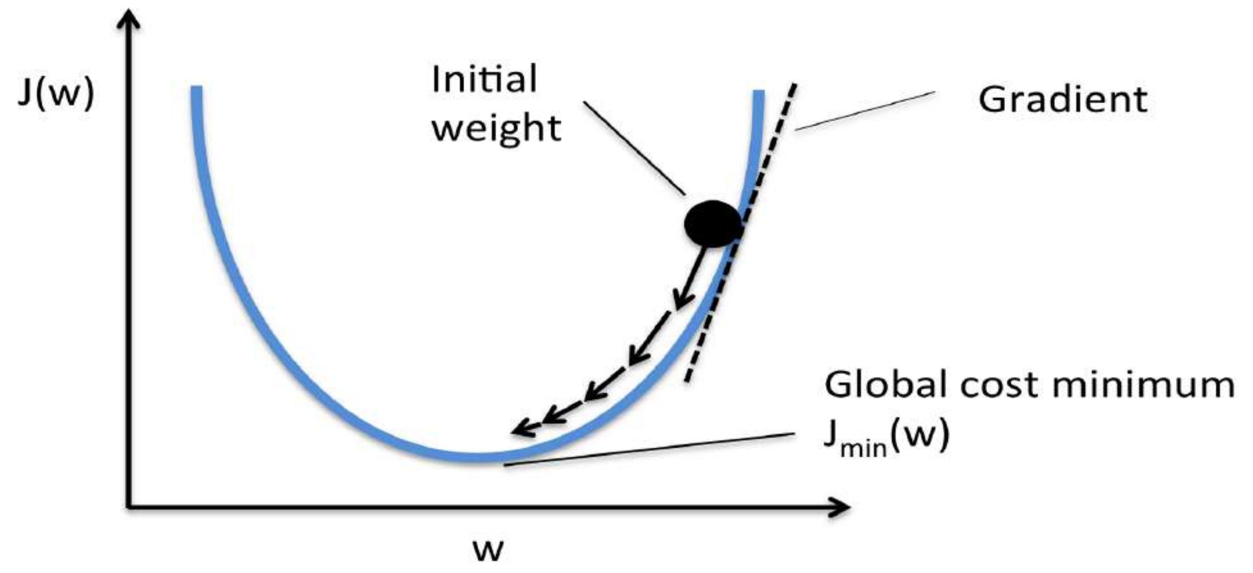


Gradient Descent Learning

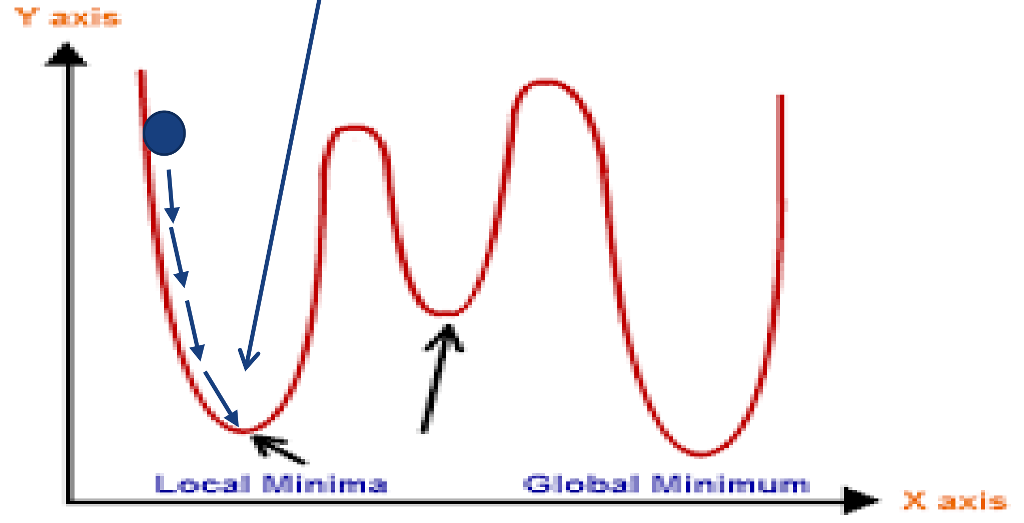
$$\Delta w_{ji}(t+1) = -\eta \frac{\partial E}{\partial w_{ji}(t)} + \alpha \Delta w_{ji}(t)$$

Leaning rate

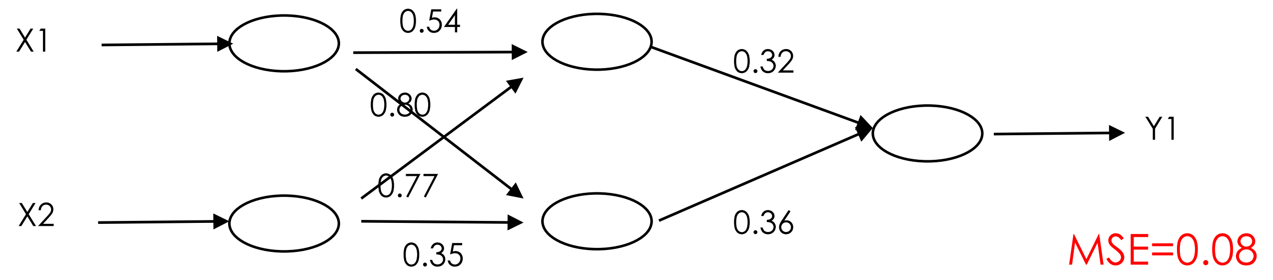
Momentum rate

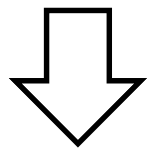


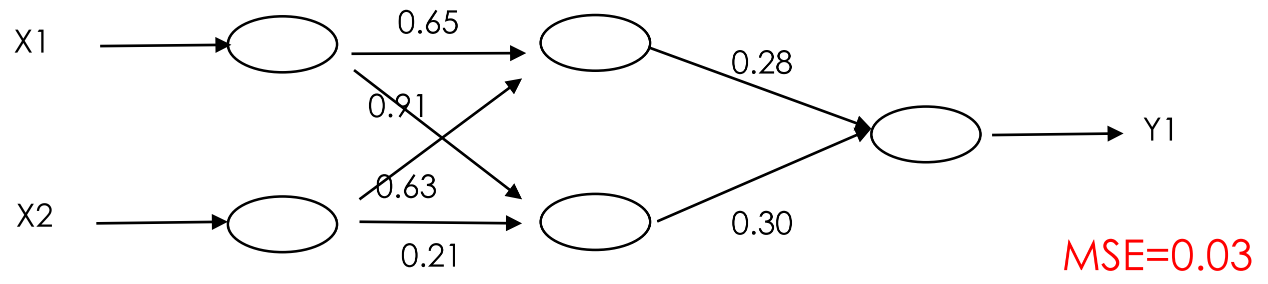
Get stuck into local minimum

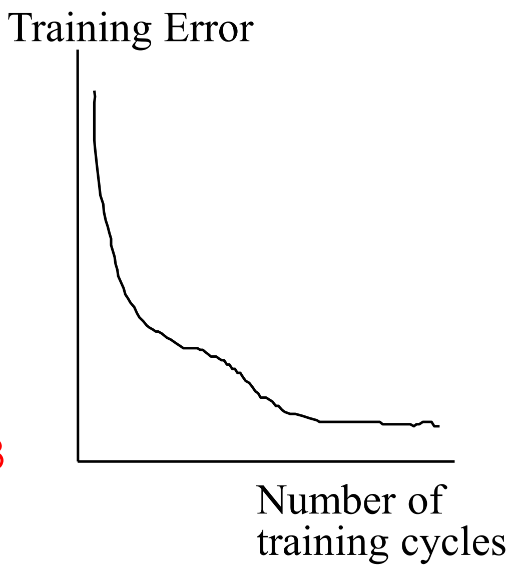


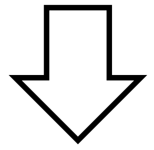
MLP Networks









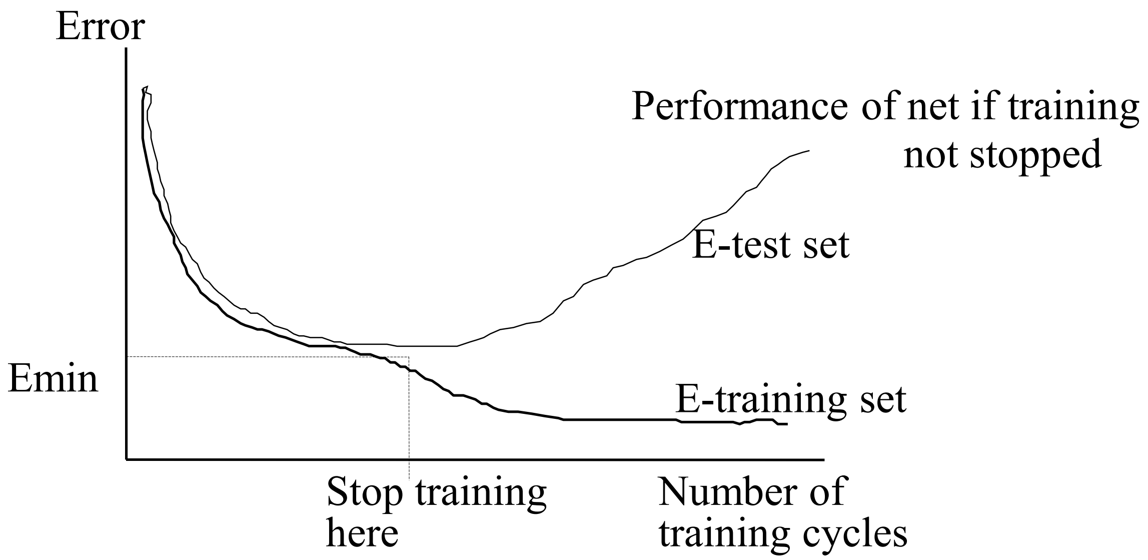


• • •   • • •

MSE=0.01

Generalization & Overtraining /Overfiting

- *Generalization* is the ability of a network to correctly classify a pattern it has not seen (not been trained on). NNs generalize when they recognize patterns not previously trained on or when they predict new outcomes from past behaviors.
- Networks can be *overtrained*. It means that they memorize the training set and are unable to generalize well.



Building NN & Pre-processing Data

- Training/test data set
  - Perform statistical analyses to support data set choices
  - Select representative training set
  - Divide training set & testing set appropriately
- Pre-processing the data
  - Data Coding
  - Data Smoothing
  - Data Transformation
    - Log $y=\log(x)$
    - Delta $\Delta x_i = x_i - x_{i-1}$
    - Normalization $\mathbf{y} = \dfrac{\mathbf{x} - \mathbf{min}(\mathbf{x})}{\mathbf{max}(\mathbf{x}) - \mathbf{min}(\mathbf{x})}$
    - Normalized Z score $z = \dfrac{x - \mu}{\sigma}$

Testing / Evaluation

- Testing the Generalization ability of a trained NN
  - **Look for good performance on a validation set and test set**
  - **Changing the training algorithm**
- The performance varies with training/ solution procedures
- Network optimization should be performed after training/testing (eliminate redundant unneeded nodes and the corresponding weights – is called *'pruning'*)
- Periodic performance testing is essential to verify model's accuracy - environmental changes can cause the data to change thereby afflicting the performance of the developed model

Applications of Neural Networks

- Image processing / Computer vision
- Natural language processing
- Data visualization
- Fault diagnosis
- Forecasting time series
- General mapping
- …

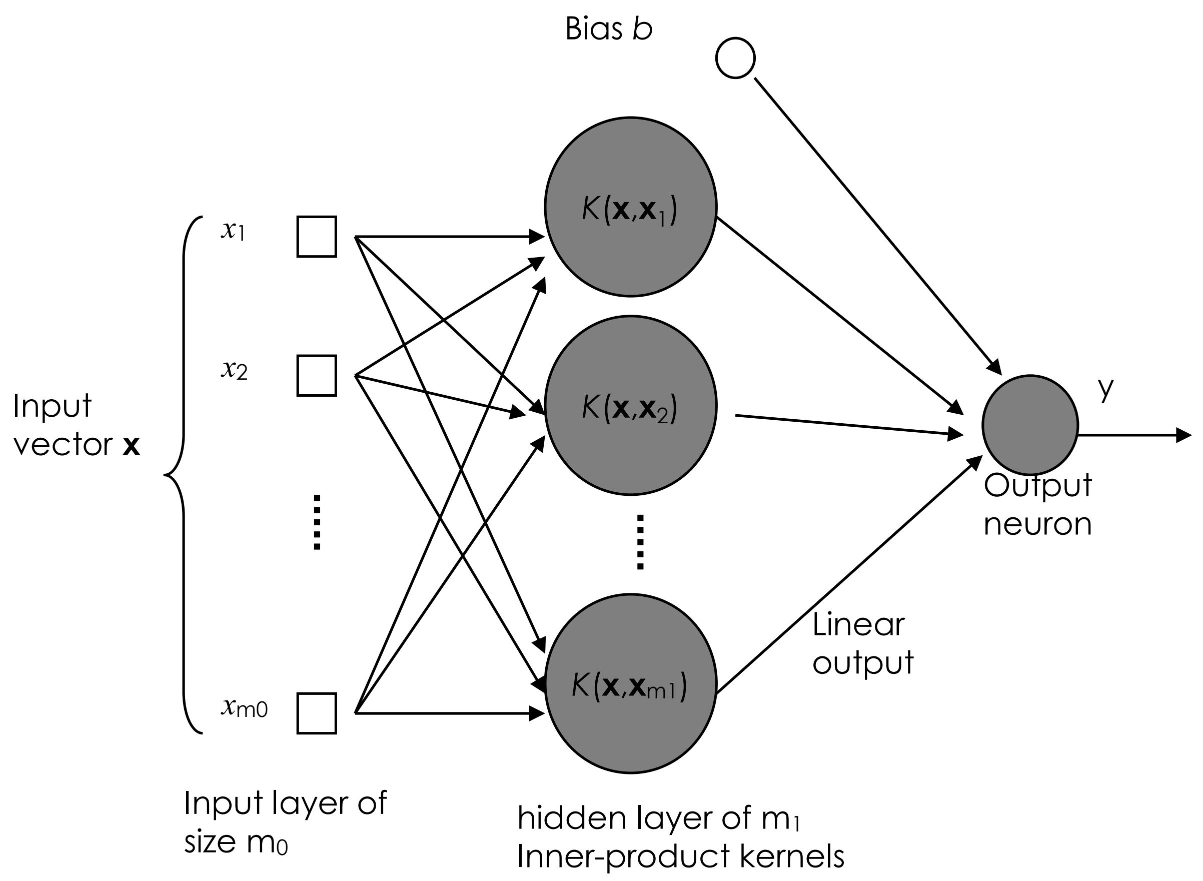
NN Modeling with Scikit-learn

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler


iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, 
random_state=0)
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)


mlp = MLPClassifier(hidden_layer_sizes=(10,10), max_iter=1000)
mlp.fit(X_train, y_train)


predictions = mlp.predict(X_test)


from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

Support Vector Machines (SVM)

- Another category of feed forward networks [Vapnik, 1992, 1995, 1998]
- SVM can be used for pattern classification and non-linear regression – but uses statistical learning theory
- General architecture of a support vector machine
  - **Input layer**
  - **Hidden layer of Inner-product kernels (fully connected with the input layer)**
  - **Output neuron**


Bias b
x_1
x_2
x_{m0}
Input vector $\mathbf{x}$
$K(\mathbf{x},\mathbf{x}_1)$
$K(\mathbf{x},\mathbf{x}_2)$
$K(\mathbf{x},\mathbf{x}_{m1})$
y
Output neuron
Linear output
Input layer of size m_0
hidden layer of m_1 Inner-product kernels

Support Vector Machines (SVM)

- For nonlinear problem, it uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating hyperplane
- SVM finds this hyperplane using support vectors ("essential" training tuples) and margins (defined by the support vectors)
- Training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, …

SVM: Optimal Hyperplane & Support Vector

- Important concepts from the theoretical background
  - *Optimal hyperplane* for separable or non-separable patterns
  - *Support vector*
- A training pattern can be represented as a *vector* from the problem space
- Consider a group of training patterns
  - Training samples: $\{(\mathbf{x}_i, y_i)\}$ $i = 1, 2, \ldots, N$

    $\mathbf{x}_i$: the input pattern for the *i*-th example

    $y_i \in \{-1,1\}$): the corresponding desired output
  - The decision surface for the separation is a hyperplane

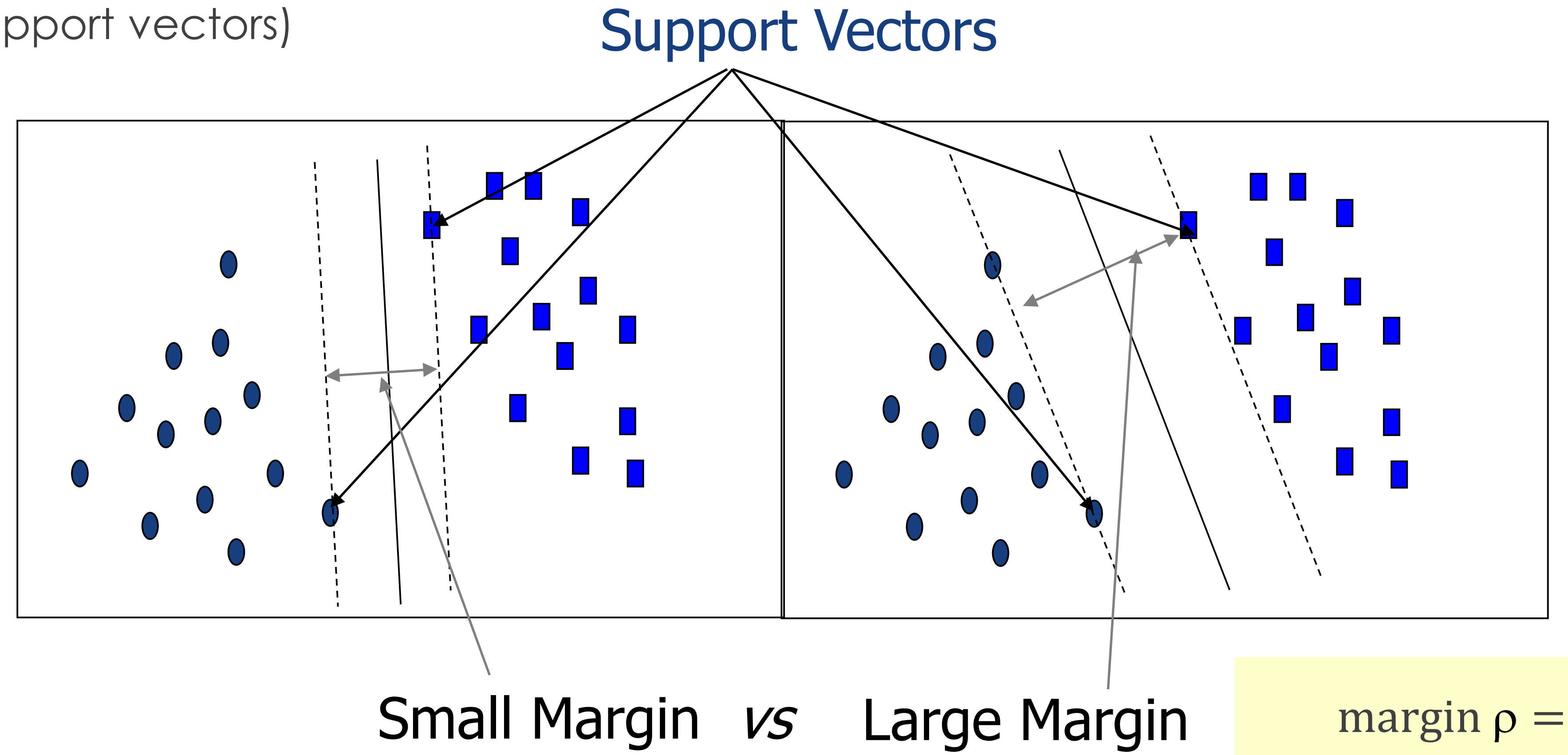
    $\mathbf{w}^T\mathbf{x} + b = 0$ (e.g. $w_1x_1 + w_2x_2 + \ldots + w_Nx_N + b = 0$)

    i.e. $\mathbf{w}^T\mathbf{x} + b \geq 0$ for $y_i = 1$

    $\mathbf{w}^T\mathbf{x} + b < 0$ for $y_i = -1$

SVM : Separation Margin & Support Vector

- *Margin of separation*
  - The separation between the decision surface hyperplane and the closest data points (support vectors)

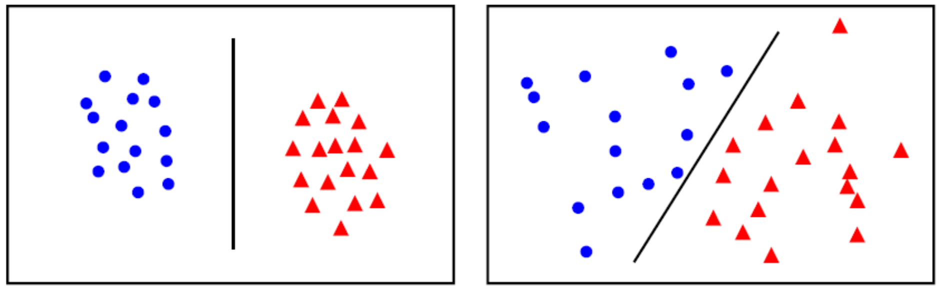


$$\text{margin } \rho = \frac{2}{||\mathbf{w}||}$$

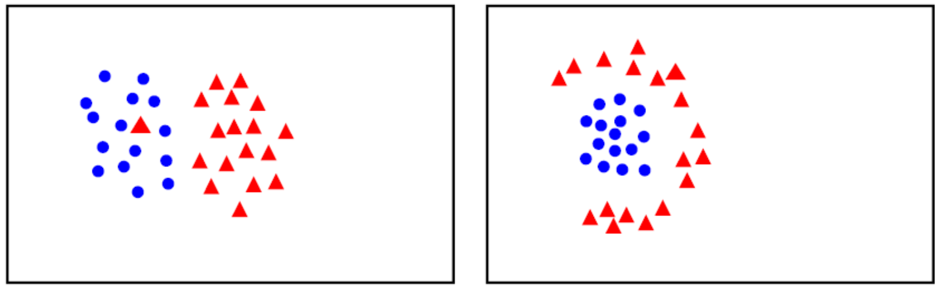
Linear Separability

- When a linear hyperplane exists to place the instances of one class on one side and those of the other class on the other side.

linearly separable

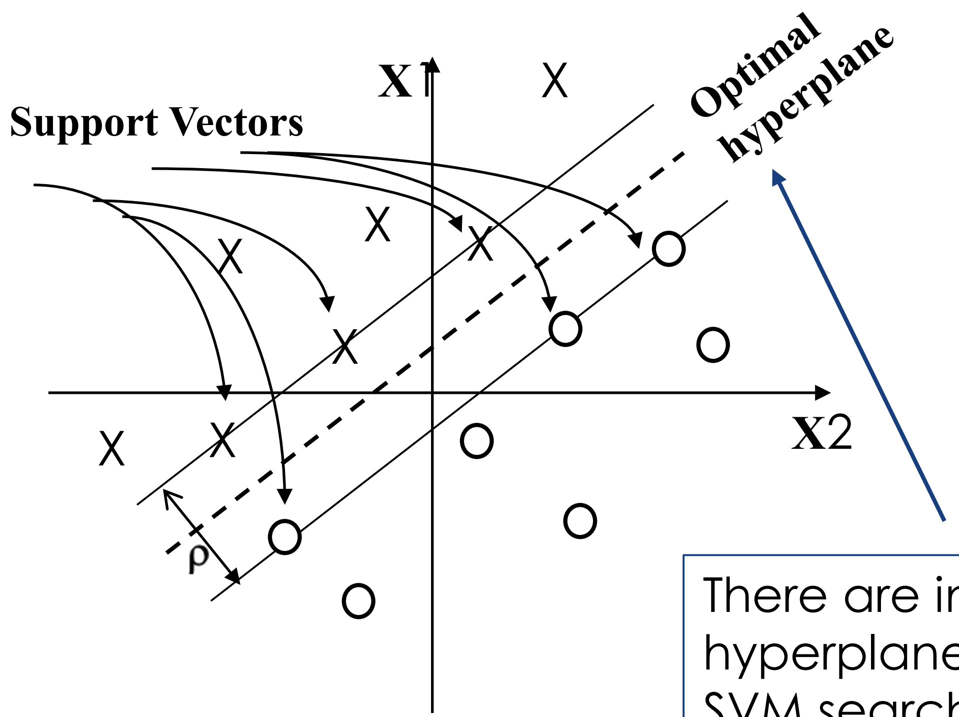


not linearly separable



Hard Margin Linear SVM : Optimal Hyperplane & Support Vector

- The goal of a support vector machine for *linearly separable patterns* is to find the particular hyper-plane for which the margin of separation ρ is maximized.
- *Support vectors*: those data points that lie closest to the decision surface and are therefore the most difficult to classify



There are infinite hyperplanes, but SVM searches for the optimal hyperplane.

Learning SVM as Optimization

maximize $\frac{2}{||\mathbf{w}||}$ $\Rightarrow$

$$\boxed{\begin{array}{lll} \text{minimize} & \frac{1}{2}\mathbf{w}^T\mathbf{w} & \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 & \forall i \end{array}}$$

where $\mathbf{w}$ satisfy
$$\begin{array}{ll} \mathbf{w}^T\mathbf{x}_i + b \geq +1 & \text{for } y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 & \text{for } y_i = -1 \end{array}$$

$Consruct\ the\ lagrangian\ function\ (primal\ problem)$:

$$J(W, b, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{l} \alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

where $a_i, i = 1, \dots, l,\ a_i \geq 0$ are Lagrange multipliers

Learning SVM as Optimization

- Dual problem:

Maximize $$Q(\boldsymbol{\alpha}) = \sum_{i}^{l} \alpha_i - \frac{1}{2} \sum_{i,j}^{l} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

Subject to: $\sum_{i=1}^{l} \alpha_i \, y_i$=0 and $\alpha_i \geq 0 \quad i = 1,2 \ldots l$

- This is a quadratic optimization problem.
- The solutions to the above dual optimization are a set of optimal α_i^* for $i = 1, \ldots, l$
  - for support vectors (SVs) $\mathbf{x}_i$, $\alpha_i^* > 0$, for non-SVs $\mathbf{x}_j$, $\alpha_j^* = 0$
  - $\boldsymbol{\alpha}^*$ determine the optimal parameters $\mathbf{w}^*$ and b^*

Learning SVM as Optimization

- Linear SVM
  - decision hypersurface is given by

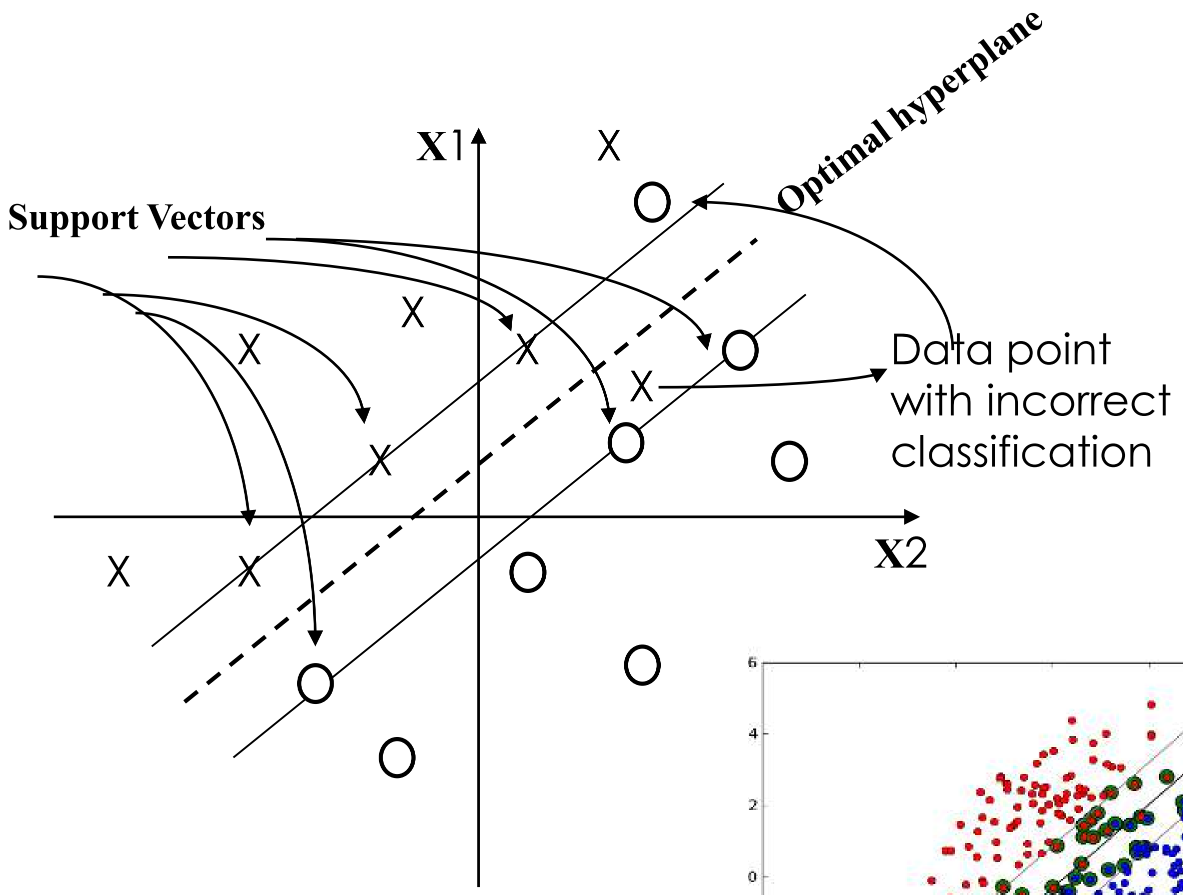
l training samples $\{(x_i, y_i)\}_l$

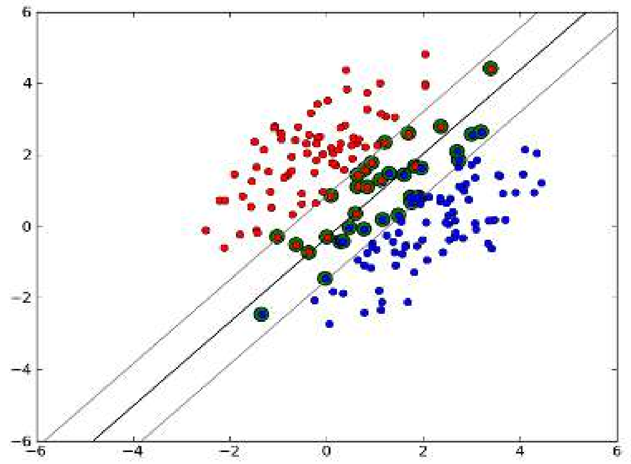
$$d(\mathbf{x}) = \boldsymbol{w}^T\mathbf{x} + b = \sum_{i=1}^{l} y_i\alpha_i \underbrace{\langle \mathbf{x}, \mathbf{x}_i \rangle}_{\text{Inner product}} + b$$

for non-SVs, $\alpha_i = 0$

Soft Margin Linear SVM: Optimal Hyperplane & Support Vector

- Given a set of *not linearly separable* training patterns, it is not possible to construct a separating hyperplane without encountering classification error.
- The goal of a support vector machine for *not linearly separable patterns* is to find an optimal hyperplane that minimizes the misclassification error, averaged over the training set.





SVM: Soft margin solution

- To classify data sets that are not linearly separable, the SVM within the linear framework is extended by introducing soft margin
  - Replace the restriction

$$\text{subject to} \qquad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

where ξ_i , called slack variables, are positive variables that indicate tolerance of misclassification.

Note that $\xi_i = 0$ if there is no error for $\mathbf{x}_i$

SVM: Soft margin solution

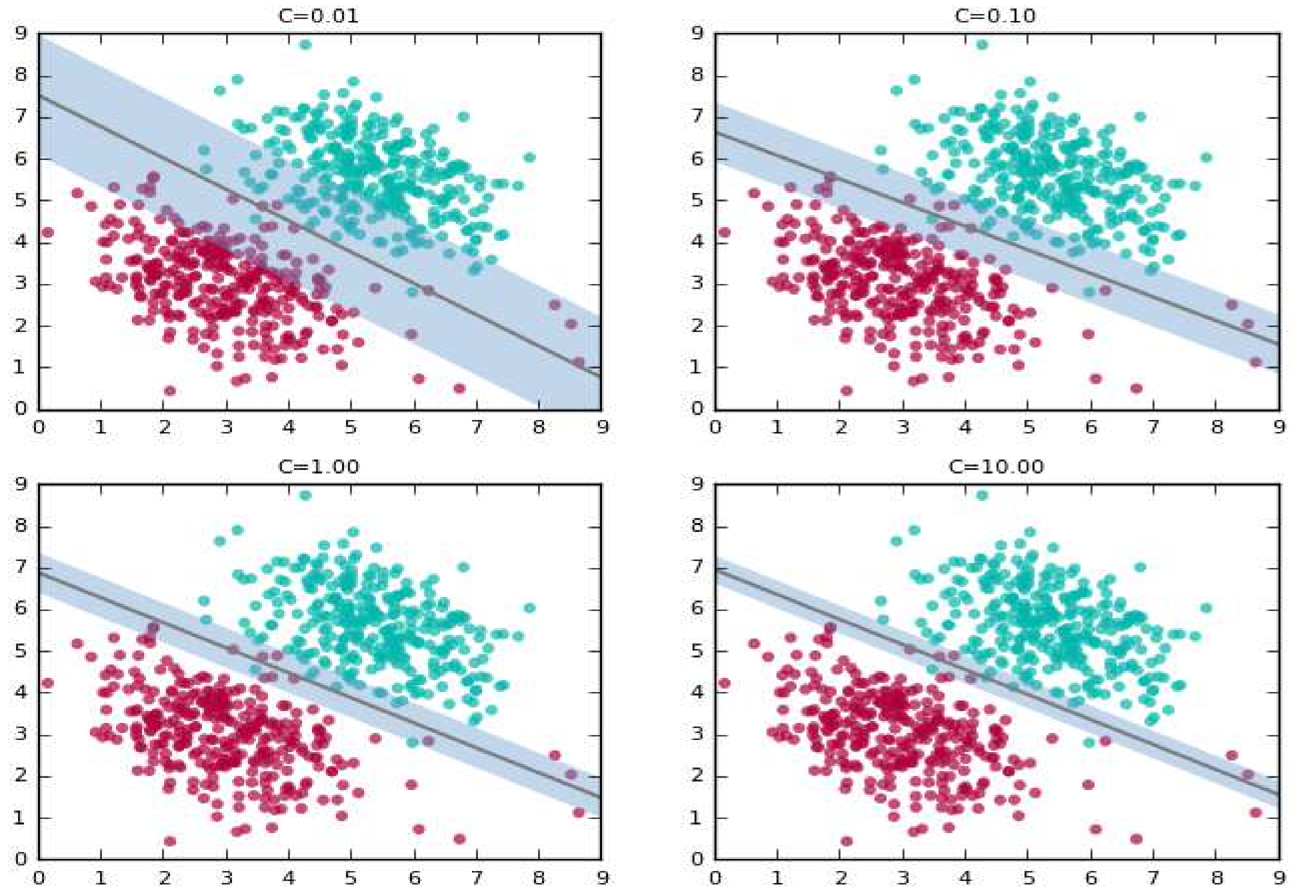
- There are optimization functions proposed for the case with soft margin, such as

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \end{array}$$

- C is a penalty parameter
  - small $C \Rightarrow$ wide margin (more tolerance)
    – many support vectors will be on the margin
  - large $C \Rightarrow$ narrow margin
    – there will be few support vectors on the margin
  - $C \rightarrow \infty$ enforces all constraints $\Rightarrow$ hard margin

SVM: Soft margin solution - C value

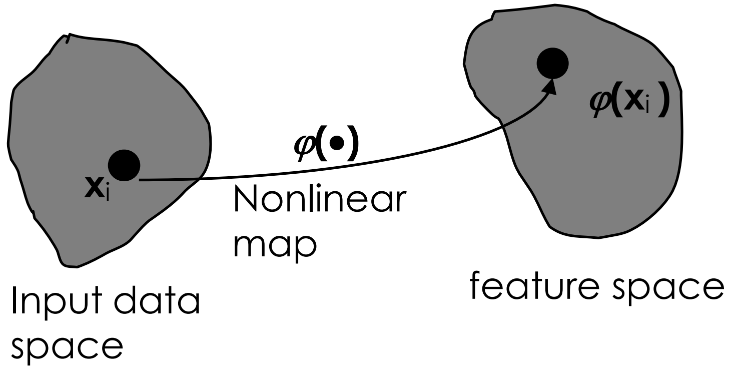
- A higher value of C implies you want lesser errors on the training data.

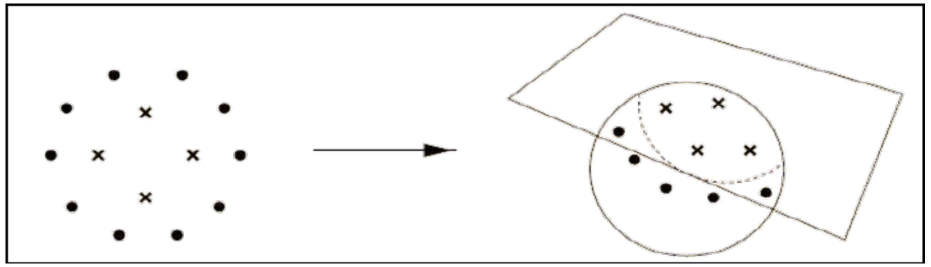


https://blog.statsbot.co/support-vector-machines-tutorial-c1618e635e93

SVM with Non–linear Kernels

- To construct a SVM for classification with an input space made up of non-linearly separable patterns
- Form Inner-product kernels
  - The multidimensional input space is transformed to a new feature space where the patterns are linearly separable with high probability, provided
    - **(a) The transformation is nonlinear**
    - **(b) The dimensionality of the feature is high enough**
  - A subset of training samples $\{x_1, x_2, \ldots x_{m1}\}$ will be used as support vectors
- Define the separating hyperplane as a linear function of vector drawn from the feature space rather than the original input space





SVM with Non–linear Kernels

- Example
  - Map the original 2-dimensional input space to a 3-dimensional feature space

$$\begin{tikzcd}
\mathrm{x} = (\mathrm{x}_1, \mathrm{x}_2) \arrow[r, "\phi(x)"'] \arrow[d] & (\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_1\mathrm{x}_2) \arrow[d] \\
\textit{Input space } \boldsymbol{X} & \textit{Feature space Z}
\end{tikzcd}$$

  - The original non-linearly separable problem becomes linearly separable in the feature space

$$\begin{tikzcd}
(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_1\mathrm{x}_2) \arrow[r, "\substack{\text{decision function} \\ y = w \cdot \phi(x) + b}"'] \arrow[d] & \mathrm{y} \in \mathrm{Y} \arrow[d] \\
\textit{Feature space Z} & \textit{Target (output) space Y}
\end{tikzcd}$$

SVM with Non–linear Kernels

- The target of learning is to achieve a minimized error of classification with decision surface
- Using dual representation we can rewrite

$$d(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

Provided in sample data

Parameters associated with input vectors in sample data, $a_i \neq 0$ for support vectors

$$d(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle + b$$

All the information the learning algorithm needs is the inner products between data points in the feature space, where all $\mathbf{x}_i$ are in the input space.

$$K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

A function that performs this direct computation of inner product is known as a kernel function, which is equivalent to the distance between **x** and **x′** measured in the higher dimensional feature space transformed by ϕ.

SVM : Typical Kernel Functions for Nonlinear Classification

- Apply a kernel function $K(X_i, X_j)$ to the original data, i.e.

$$K(X_i, X_j) = \Phi(X_i)\ \Phi(X_j)$$

- Typical Kernel Functions

Polynomial kernel of degree h : $K(\boldsymbol{X_i}, \boldsymbol{X_j}) = (\boldsymbol{X_i} \cdot \boldsymbol{X_j} + 1)^h$

Gaussian radial basis function kernel : $K(\boldsymbol{X_i}, \boldsymbol{X_j}) = e^{-\|\boldsymbol{X_i} - \boldsymbol{X_j}\|^2 / 2\sigma^2}$

Sigmoid kernel : $K(\boldsymbol{X_i}, \boldsymbol{X_j}) = \tanh(\kappa \boldsymbol{X_i} \cdot \boldsymbol{X_j} - \delta)$

SVM Example

- Objective: Classification for 1-D data
- Suppose we have 5 training data points
  - $x_1=1$, $x_2=2$, $x_3=4$, $x_4=5$, $x_5=6$, with 1, 2, 6 as class A and 4, 5 as class B $\Rightarrow$ $y_1=1$, $y_2=1$, $y_3=-1$, $y_4=-1$, $y_5=1$
- We use the polynomial kernel $K(x_i, x_j) = (x_i . x_j + 1)^2$ and C is set to 100. We need to find α_i ($i=1, \ldots, 5$) by

$$\text{max.} \quad \sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^{5} \alpha_i y_i = 0$$

SVM Example

- After solving optimization problem, we get
  - α_1=0, α_2=2.5, α_3=0, α_4=7.333, α_5=4.833
  - The support vectors are {x_2=2, x_4=5, x_5=6}
- For a new point z, the discriminant function is

$$\begin{aligned} & f(z) \\ &= 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + 4.833(1)(6z+1)^2 + b \\ &= 0.6667z^2 - 5.333z + b \end{aligned}$$

- *b* is solved by solving f(2)=1 or by f(5)=-1 or by f(6)=1, all three give b=9

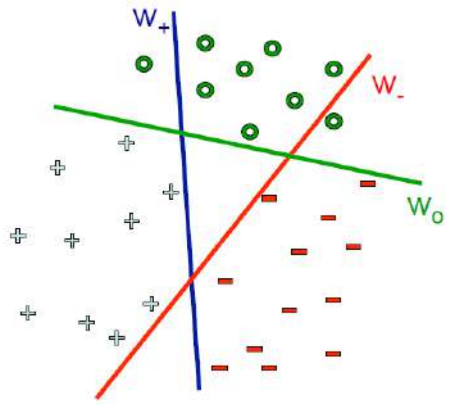
$$\boxed{f(z) = 0.6667z^2 - 5.333z + 9}$$

SVM in Practice

- Prepare the dataset
- Select the kernel function to use
- Select the parameter of the kernel function and the value of C
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the a_i
- Test data can be classified using the a_i and the support vectors

Multi-class SVM Classifier

- One vs. others
  - Training: Learn an SVM for each vs. the others
  - Testing: Apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value



- Learn 3 classifiers:
  - - vs. {o,+}, weights w_-
  - + vs. {o,-}, weights w_+
  - o vs. {+,-}, weights w_o
- Predict label using:

$$\hat{y} \leftarrow \arg\max_k \; w_k \cdot x + b_k$$

- One vs. one
  - Training: Learn an SVM for each pair of classes
  - Testing: Major voting from each learned SVM

Applications of SVM

- SVMs have been widely applied in
  - Bioinfomatics
  - Machine Vision
  - Text Categorization
  - Handwritten Character Recognition
  - ……

SVM with Scikit-learn

```
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC


iris = load_iris()

X = iris.data

y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
```

```
scaler = StandardScaler()

scaler.fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)


svm=SVC(kernel="rbf", gamma=5, C=1)

svm.fit(X_train, y_train)


predictions = svm.predict(X_test)


from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test,predictions))

print(classification_report(y_test,predictions))
```

Support Vector Machines: Summary

- The SVM is an elegant and highly principled learning method for the design of a feedforward network with a single hidden layer of nonlinear units
- Design hinges on the extraction of a subset of the training data that serves as support vectors and therefore represents a stable characteristic of the data
- Learning in SVM
  - Learning algorithm operates only in a batch mode
  - The near-to-perfect classification performance is achieved at the cost of a significant demand on computational complexity
- The complexity of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- An SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

2.2
Pattern Recognition Workshop 2

Workshop 2

- Open the iPython notebook provided for workshop 2.
- You will build decision tree, neural network and SVM models in this workshop.
- As you go through the notebook, make sure you understand how each different model is built. (you can save notes as markdown in the notebook).
- Compare the performance of these models.
- Experiment with different parameter settings.
- You may try with your own datasets.
- Save your notebook with the cell output and upload it to LumiNUS.