Data Mining Algorithms: Prediction

—

Department of Computer Science and Engineering
Kathmandu University

k-Nearest Neighbour (kNN)

- k-Nearest Neighbor (kNN) classifier determines the class of an example based on the labels of its neighbors belonging to the training set.
- It is a lazy learner. It does not produce a model.
  - When given a training tuple, it simply stores it, and waits until it is given a test tuple.
  - Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples.
- Uses k “closest” points (nearest neighbors) for performing classification.
- Basic idea: If it walks like a duck, quacks like a duck, then it’s probably a duck.

k-Nearest Neighbour (kNN)

Requires three things

- **Training set:** the set of labeled instances
- **A Distance Metric:** to compute distance between records
  - A function f(x, y) may be used as a distance function if four conditions are met:
    i. $f(x, y) \geq 0$
    ii. $f(x, x) = 0$
    iii. $f(x, y) = f(y, x)$
    iv. $f(x, y) \leq f(x, z) + f(z, y)$.
- Some value of **k**, the number of nearest neighbors to retrieve / consider

k-Nearest Neighbour (kNN)

To classify an unknown record

1. Compute distance to other training instances
2. Identify k neighbors with the shortest distance (nearest neighbors)
3. Use class labels of the nearest neighbors to determine the class label of the unknown record (e.g., by taking majority vote, by weighing the votes according to distance)

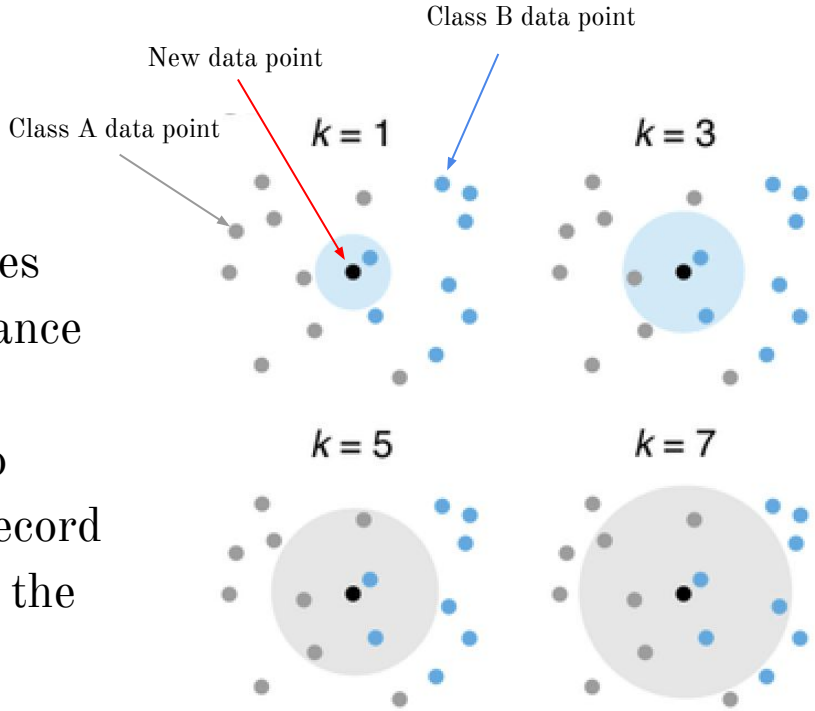


Image source: Bzdok et al., 2018

Distance metrics

- A function f(x, y) may be used as a distance function if following conditions are met:
  a. Non-negativity: $f(x, y) \geq 0$ and $f(x, x) = 0$
  b. Symmetry: $f(x, y) = f(y, x)$
  c. Triangular inequality: $f(x, y) \leq f(x, z) + f(z, y)$
- Commonly used distance metrics:
  a. Euclidean distance
  b. Manhattan distance
  c. Cosine distance

Distance metrics

Euclidean distance $d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$

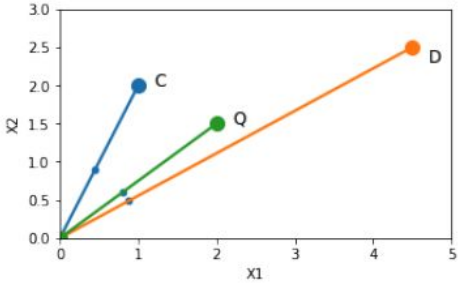
Manhattan distance $d(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{n} |x_k - y_k|$

Image source: https://www.translatorscafe.com/unit-converter/en-US/calculator/two-points-distance/

Cosine distance = 1 - CosineSimilarity

Cosine similarity measures the angle between two vectors.

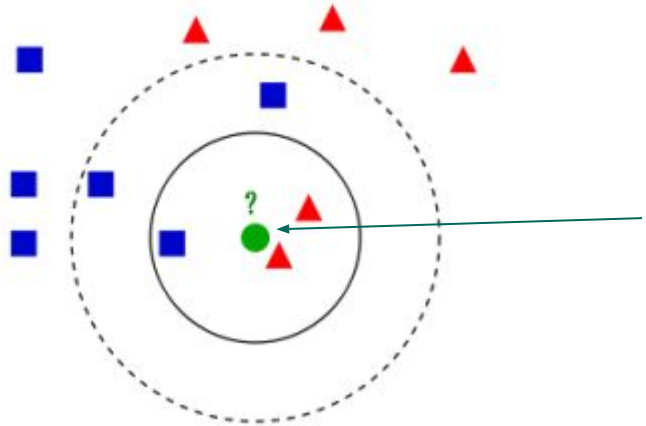
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$



While C would be the closer example to Q based on Euclidean distance, D is closer to Q based on cosine similarity. (Cunningham & Delany, 2020)

k-Nearest Neighbor (kNN)

kNN is very sensitive to the value of parameter k.



Classified as red when k = 3 but classified as blue when k = 5

Bayesian classifiers

- Bayesian classifiers can predict class membership probabilities such as the probability that a given instance belongs to a particular class.
- Bayesian classification is based on Bayes' Theorem.
  - Let X be a data tuple. For classification problems, we want to determine the probability that the tuple X belongs to class C, given that we know the attribute description of X, i.e., P(C|X).
  - According to the Bayes theorem

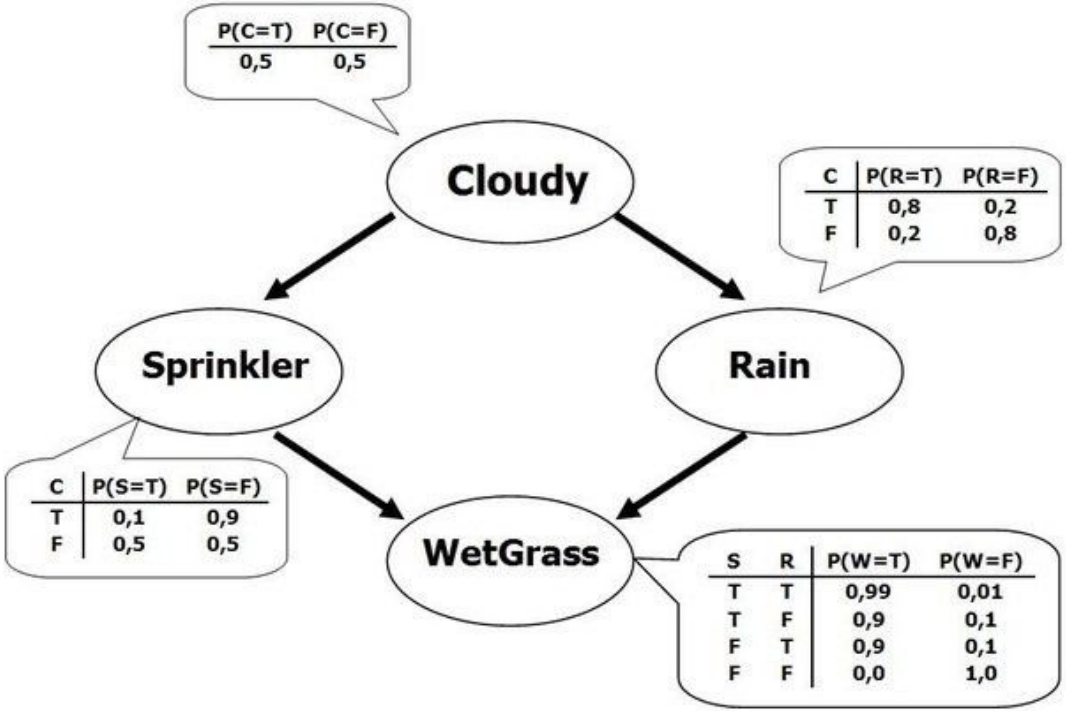
$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

where
P(C|X) is the posterior probability of C conditioned to X, and
P(C) is the prior probability of C.

Bayesian Networks

- A Bayesian network (aka Bayesian belief network), provides a graphical representation of the probabilistic relationships among a set of random variables.
- Two key elements of a Bayesian network
  a. A directed acyclic graph (DAG) encoding the dependence relationships among a set of variables.
     - Each node represents a variable.
     - Each arc asserts the dependence relationship between the pair of variables.
  b. A probability table associated with each node to its immediate parent nodes.
     - If a node X has no parent, then the table contains only the prior probability P(X).
     - If a node X has only one parent Y, then the table contains the conditional probability P(X|Y).
     - If a node X has multiple parents $\{Y_1, Y_2, \dots, Y_n\}$, then the table contains the conditional probability $P(X|Y_1, Y_2, \dots, Y_n)$.

Bayesian Networks



- Rain depends probabilistically on Cloudy. WetGrass depends probabilistically on Rain and so on.
- If we know whether it rained and whether the sprinkler was used, then WetGrass and Cloudy become independent to each other.

Reminders

Independence

- A and B are independent iff
  - $P(A, B) = P(A)\,P(B)$
  - $P(A|B) = P(A)$
  - $P(B|A) = P(B)$

Conditional Independence

- A and B are conditionally independent to C $(A \perp\!\!\!\perp B \mid C)$ iff $P(A \mid B, C) = P(A \mid C)$

Chain rule

$$P(X_n, \ldots, X_1) = P(X_n | X_{n-1}, \ldots, X_1) \cdot P(X_{n-1}, \ldots, X_1)$$

$$P(A_1, A_2, ..., A_n) = P(A_1)\,P(A_2|A_1)\,P(A_3|A_1, A_2) \ldots P(A_n|A_1, A_2, ..., A_n)$$

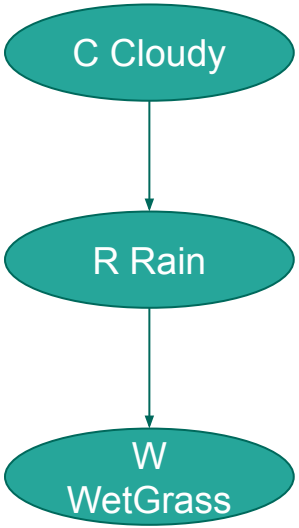
Bayesian Networks

- An important property of the Bayesian network is that a node in a Bayesian network is conditionally independent of its non-descendants, if its parents are known.
- This allows the network to provide a complete representation of the existing joint probability distribution with the following equation:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | Parents(Y_i))$$

where $P(x_1,..., x_n)$ is the probability of a particular combination of values of X, X being a data tuple described by the variables or attributes $Y_1,.. Y_n$, and the values for $P(x_i|Parents(Y_i))$ correspond to the entries in the CPT for Y_i.

Bayesian Networks

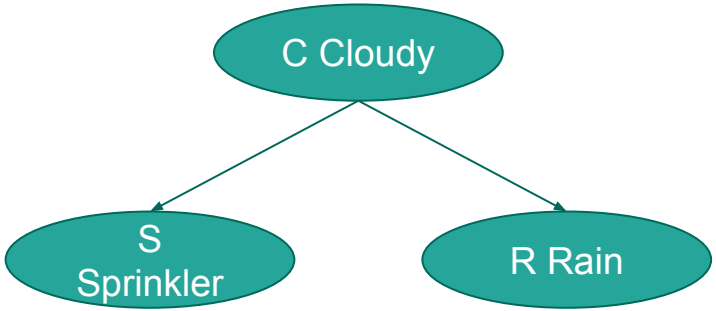


C and W are dependent.

C and W are conditionally independent to R, i.e., if R is known, W will not give any new information about C.

$$P(W \mid C, R) = P(W \mid R) = P(W \mid \text{Parents}(W))$$

Bayesian Networks

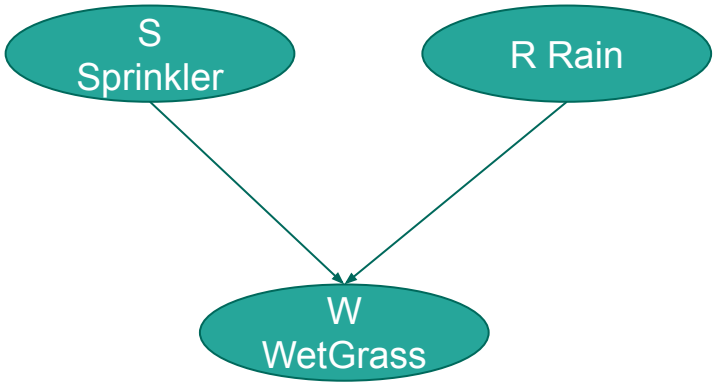


S and R are dependent.

S and R are conditionally independent to C, i.e., if C is known, R will not give any new information about S.

$$P(S \mid C, R) = P(S \mid C) = P(S \mid \text{Parents}(S))$$

Bayesian Networks

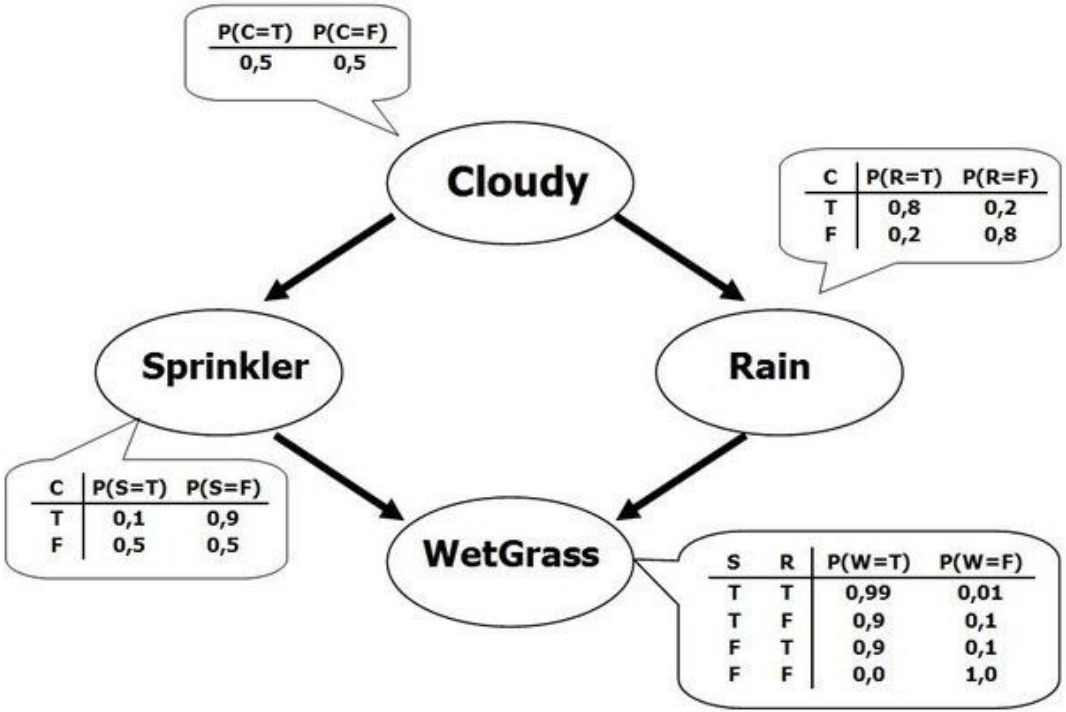


S and R are independent.

S and R are conditionally dependent to W, i.e., if W is known, R will give some new information about S.

$P(W \mid S, R) = P(W \mid \text{Parents}(W))$

Bayesian Networks



$$P(C, R, S, W)$$
$$= P(C)\ P(R|C)\ P(S|R,C)\ P(W|S,R,C)$$

$$= P(C)\ P(R|C)\ P(S|C)\ P(W|S,R)$$

Inference

Finding the probability of some assignment of a subset of the variables (x) given assignments of other variables (our evidence, e), i.e. finding P(x | e)

Example: P(Cloudy = T | WetGrass = T) = ?

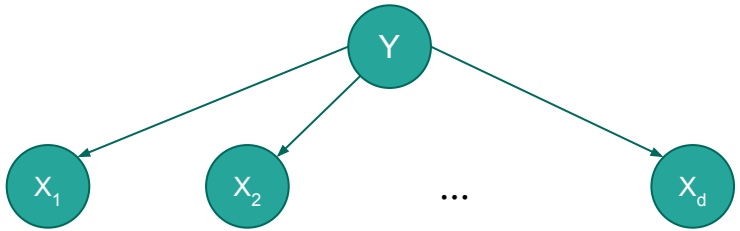
- Exact inference algorithms
  - Variable elimination, message passing, junction tree etc.
- Approximate inference algorithms
  - Sampling, Monte Carlo Markov Chain (MCMC) simulation, Loopy belief propagation etc.

Naïve Bayes Classifier

A naïve Bayes classifier estimates the class-conditional probability by assuming that the attributes are conditionally independent, given the class label.

$$P(\mathbf{X}|Y) = \prod_{i=1}^{d} P(X_i|Y)$$

where $\mathbf{X} = \{X_1, X_2, ..., X_d\}$ is the attribute set containing d attributes and Y denote the class variable.



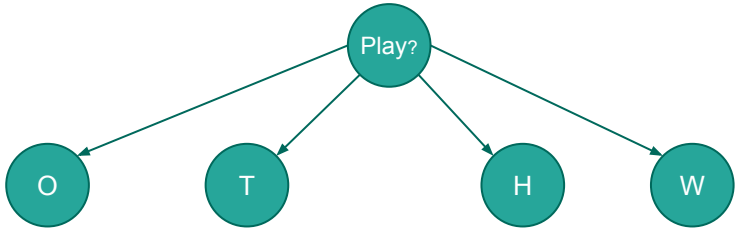
Naïve Bayes Classifier

To classify a record, the naive Bayes classifier computes the posterior probability for each class Y:

$$P(Y|\mathbf{X}) = \frac{P(Y)\prod_{i=1}^{d} P(X_i|Y)}{P(\mathbf{X})}$$

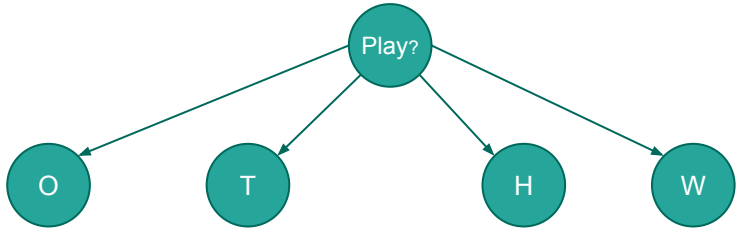
Naïve Bayes Classifier: Example

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



Naïve Bayes Classifier: Example

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

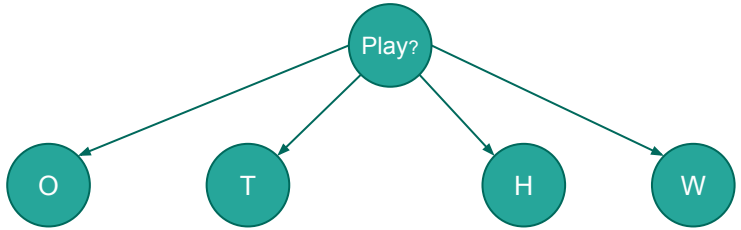


$$P(\text{Outlook} = \text{sunny} \mid \text{Play} = \text{Yes}) = ?$$
$$P(\text{Outlook} = \text{rain} \mid \text{Play} = \text{Yes}) = ?$$
$$P(\text{Outlook} = \text{overcast} \mid \text{Play} = \text{Yes}) = ?$$

$$P(\text{Outlook} = \text{sunny} \mid \text{Play} = \text{No}) = ?$$
$$P(\text{Outlook} = \text{rain} \mid \text{Play} = \text{No}) = ?$$
$$P(\text{Outlook} = \text{overcast} \mid \text{Play} = \text{No}) = ?$$

Naïve Bayes Classifier: Example

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

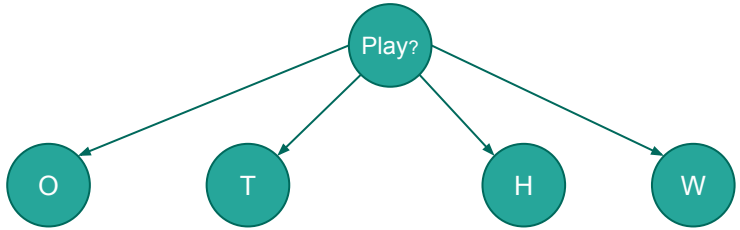


$$P(\text{Outlook} = \text{sunny} \mid \text{Play} = \text{Yes}) = 2/9$$
$$P(\text{Outlook} = \text{rain} \mid \text{Play} = \text{Yes}) = 3/9$$
$$P(\text{Outlook} = \text{overcast} \mid \text{Play} = \text{Yes}) = 4/9$$

$$P(\text{Outlook} = \text{sunny} \mid \text{Play} = \text{No}) = 3/5$$
$$P(\text{Outlook} = \text{rain} \mid \text{Play} = \text{No}) = 2/5$$
$$P(\text{Outlook} = \text{overcast} \mid \text{Play} = \text{No}) = 0$$

Naïve Bayes Classifier: Example

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

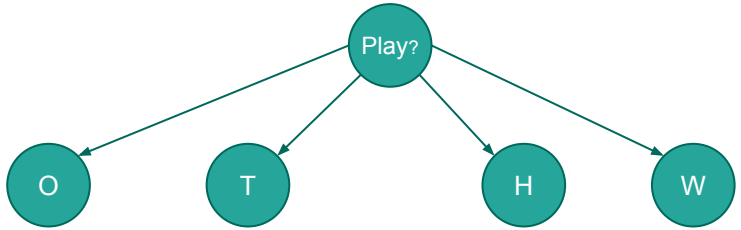


$$P(\text{Temperature} = \text{hot} \mid \text{Play} = \text{Yes}) = 2/9$$
$$P(\text{Temperature} = \text{mild} \mid \text{Play} = \text{Yes}) = 4/9$$
$$P(\text{Temperature} = \text{cool} \mid \text{Play} = \text{Yes}) = 3/9$$

$$P(\text{Temperature} = \text{hot} \mid \text{Play} = \text{No}) = 2/5$$
$$P(\text{Temperature} = \text{mild} \mid \text{Play} = \text{No}) = 2/5$$
$$P(\text{Temperature} = \text{cool} \mid \text{Play} = \text{No}) = 1/5$$

Naïve Bayes Classifier: Example

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

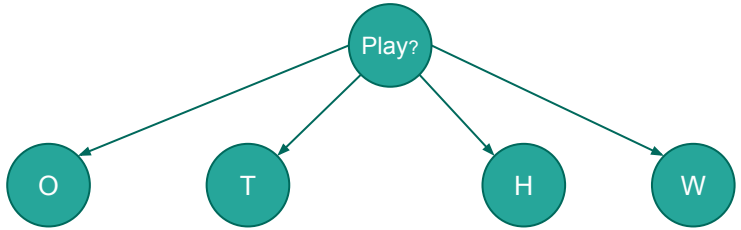


$$P(\text{Humidity} = \text{high} \mid \text{Play} = \text{Yes}) = 3/9$$
$$P(\text{Humidity} = \text{normal} \mid \text{Play} = \text{Yes}) = 6/9$$

$$P(\text{Humidity} = \text{high} \mid \text{Play} = \text{No}) = 4/5$$
$$P(\text{Humidity} = \text{normal} \mid \text{Play} = \text{No}) = 1/5$$

Naïve Bayes Classifier: Example

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

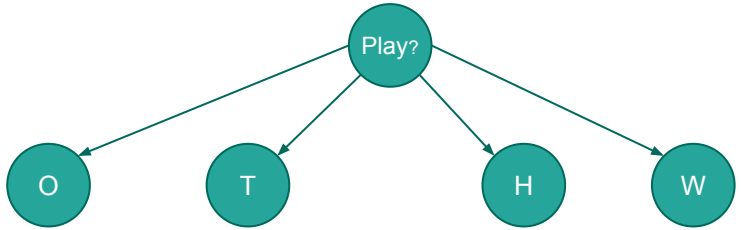


$$P(\text{Windy} = \text{false} \mid \text{Play} = \text{Yes}) = 6/9$$
$$P(\text{Windy} = \text{true} \mid \text{Play} = \text{Yes}) = 3/9$$

$$P(\text{Windy} = \text{false} \mid \text{Play} = \text{No}) = 2/5$$
$$P(\text{Windy} = \text{true} \mid \text{Play} = \text{No}) = 3/5$$

Naïve Bayes Classifier: Example

Outlook	Temperature	Humidity	Windy	Play?
overcast	mild	normal	true	?



To predict the class label of this test record,
we need to compute the posterior probabilities
$P(Play = No \mid X)$ and $P(Play = Yes \mid X)$,
where X = {Outlook = overcast, Temperature = mild, Humidity = normal, Windy = true}

$$P(Play = No \mid X) \propto P(Play = No) \times P(Outlook = overcast \mid Play = No) \times$$
$$P(Temperature = mild \mid Play = No) \times$$
$$P(Humidity = normal \mid Play = No) \times$$
$$P(Windy = true \mid Play = No)$$

Linear models

Decision trees and rules work most naturally with nominal attributes.

They can be extended to numeric attributes

- By incorporating numeric-value tests directly into the decision tree or rule induction scheme, or
- By pre-discretizing numeric attributes into nominal ones.

However, there are methods that work most naturally with numeric attributes.

- Regression
- Perceptron

Linear models

- Can be understood **in terms of lines and planes**.
- Are **parametric**, i.e. they have a fixed form with a small number of numeric parameters that need to be learned from data.
- Are commonly **stable**, i.e., small variations in the training data have only limited impact on the learned model.
- Are **less likely to overfit** the training data than some other models, largely because they have relatively few parameters. However, they sometimes lead to **underfitting**.

Linear regression

Given: Data with d attributes and 1 target-variable (real number)

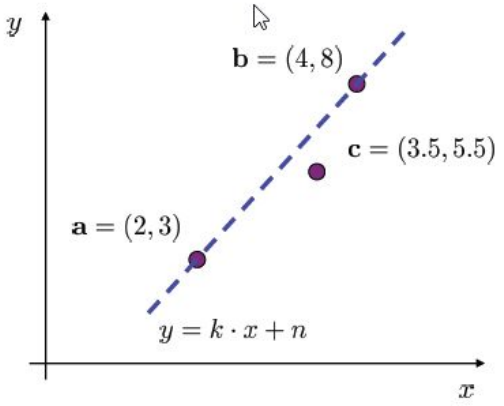
$$\{(X_i, y_i)|i = 1, 2, \ldots, n\}$$

Where each X_i corresponds to the set of attributes of the ith observation (aka **explanatory variables**) and y_i corresponds to the **target** (or response).

Objective: Fit the following linear model to the observed data

$$f(x) = w_0 + w_1 x_1 + \cdots + w_n x_n$$

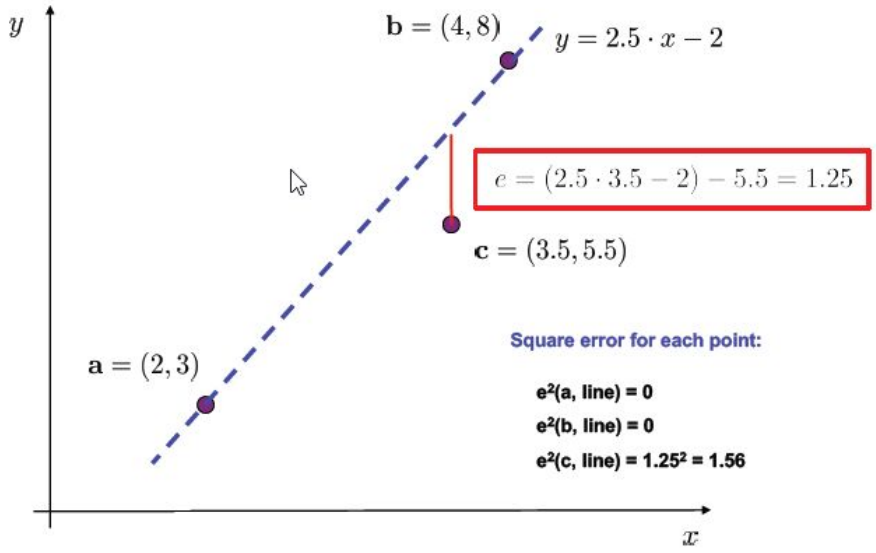
Where w_0, w_1, ..., w_n are called the r**egression coefficients.**



Linear regression: The least-squares method

A standard approach for linear regression is to apply the method of least squares, which attempts to find the parameters w_0, w_1, ..., w_n that minimize the **sum of the squared error (SSE)** (aka **residual sum of squares**).

$$SSE = \sum_{i=1}^{n} [y_i - f(X_i)]^2$$



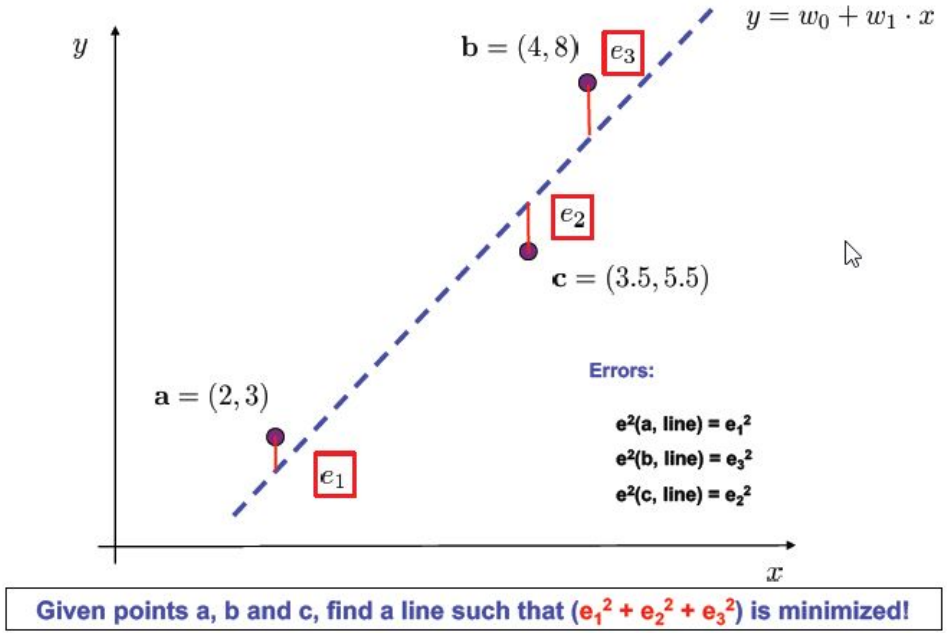
Linear regression: The least-squares method

Solving for the parameters, we get

$$W = (X^T X)^{-1} X^T Y$$

where

$$X = \begin{pmatrix} 1 & X_1 \\ 1 & X_2 \\ \dots & \\ 1 & X_n \end{pmatrix} \quad W = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{pmatrix} \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$



Ordinary least square linear regression

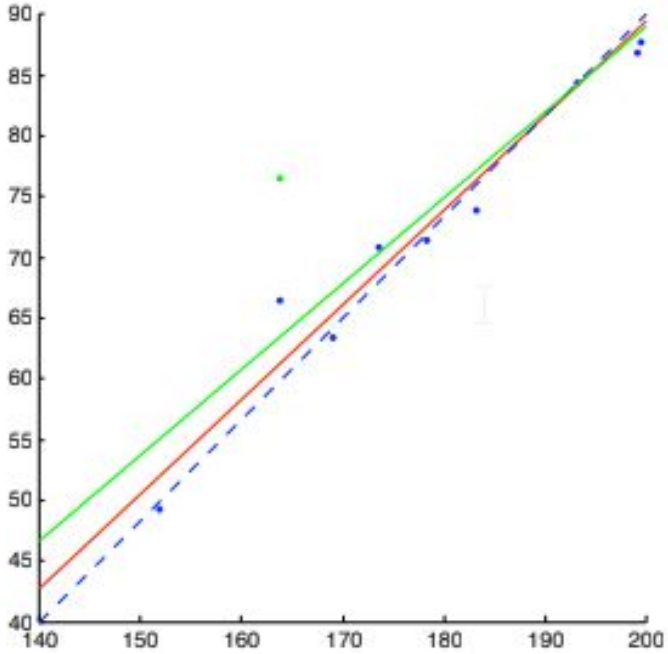
The effect of outliers

OLS linear regression is **susceptible to outliers**

Despite this sensitivity to outliers, the least-squares method usually works surprisingly well.

In the figure:

Green point = outlier
Red line = Regression line without the outlier
Green point = Regression line with the outlier



Linear regression: Variants

Regularized regression

Least-squares regression can become unstable (i.e., highly dependent on the training data), which can lead to overfit.

To avoid overfitting, a regularization term can be introduced so as to ensure the weights are, on average, small in magnitude (shrinkage).

Ridge regression: minimize the absolute sum of the coefficients

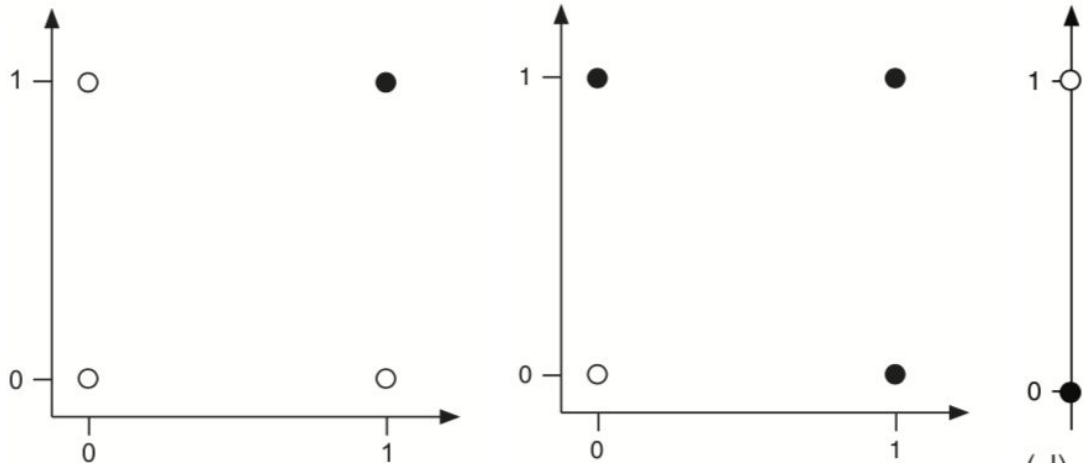
$$w^* = \arg\min_w (Y - XW)^T (Y - XW) + \lambda \|W\|^2$$

Lasso (Least absolute shrinkage and selection provider): minimize the squared absolute sum of the coefficients $w^* = \arg\min_w (Y - XW)^T (Y - XW) + \lambda \sum_i |W_i|$

Perceptron

A perceptron is a single-layer Neural Network.

It achieves perfect separation on **linearly separable** data.



Perceptron

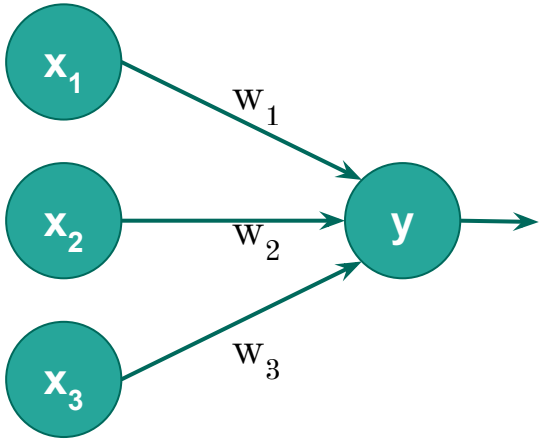
A perceptron is a single-layer Neural Network.

It achieves perfect separation on **linearly separable** data.

Input nodes (or units) are connected (typically fully) to a node (or multiple nodes) in the next layer.

A node in the next layer takes a weighted sum of all its inputs

$$\sum_i w_i x_i$$



Perceptron

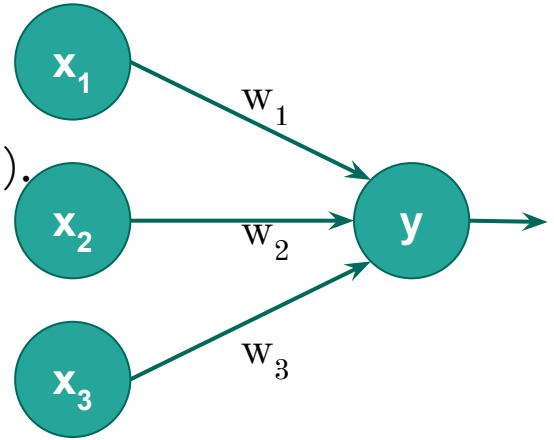
The output node has a "threshold" t.

Rule:

If summed input $\geq$ t, then it "fires" (output y = 1).

Else (i.e. summed input $<$ t) it doesn't fire (output y = 0).

The perceptron is simply s**eparating the input into 2 categories**, those that cause a fire, and those that don't.



Perceptron

The output of a perceptron is

$$\hat{y} = sign(w_n x_n + w_{n-1} x_{n-1} + \cdots + w_2 x_2 + w_1 x_1 - t)$$

Where the sign function, which acts as an activation function, outputs a value +1 if its argument is positive and -1 if its argument is negative.

In a more compact form,

$$\hat{y} = sign(w_n x_n + w_{n-1} x_{n-1} + \cdots + w_2 x_2 + w_1 x_1 + w_0 x_0) = sign(\mathbf{w}.\mathbf{x})$$

Where $w_0 = -t$, $x_0 = 1$ and **w.x** is the dot product between the weight vector **w** and the input attribute vector **x**.

Perceptron learning

Task: Learn the weights (and the threshold)

The perceptron iterates over the training set, updating the weight vector every time it encounters an incorrectly classified example.

Algorithm 7.1: Perceptron(D, η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates;
learning rate η.
Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
1  w ← 0 ;                          // Other initialisations of the weight vector are possible
2  converged ← false;
3  while converged = false do
4      converged ← true;
5      for i = 1 to |D| do
6          if y_i w · x_i ≤ 0        // i.e., ŷ_i ≠ y_i
7          then
8              w ← w + η y_i x_i;
9              converged ← false;    // We changed w so haven't converged yet
10         end
11     end
12 end
```

Perceptron learning

If y = +1 and w.x = -1, we need to increase the value of the predicted output by increasing the weights of all links with positive inputs and decreasing the weights of all links with the negative inputs.

Algorithm 7.1: Perceptron(D, η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates;
learning rate η.

Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
1  w ← 0;                          // Other initialisations of the weight vector are possible
2  converged ← false;
3  while converged = false do
4      converged ← true;
5      for i = 1 to |D| do
6          if y_i w · x_i ≤ 0          // i.e., ŷ_i ≠ y_i
7          then
8              w ← w + η y_i x_i;
9              converged ← false;      // We changed w so haven't converged yet
10         end
11     end
12 end
```

Perceptron

New weight

$$\mathbf{w}' = \mathbf{w} + \eta y^{(i)} \mathbf{x}^{(i)}$$

where
$x^{(i)}$ is the misclassified example,
$y^{(i)}$ is the class of $x^{(i)}$ in the training data, and
η is the **learning rate** (value between 0 and 1), which controls the amount of adjustments made in each iteration.

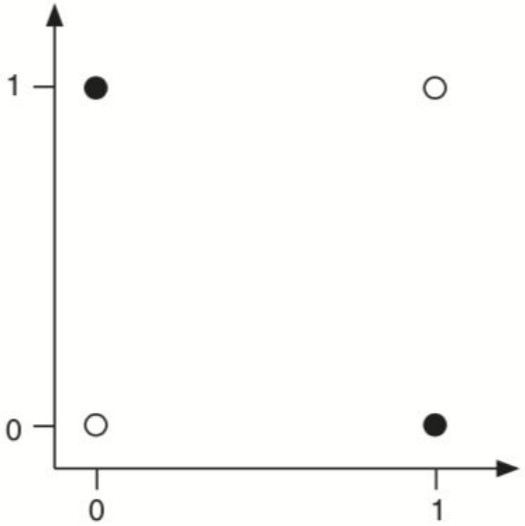
Algorithm 7.1: Perceptron(D, η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates;
learning rate η.
Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
1  w ← 0;                          // Other initialisations of the weight vector are possible
2  converged ← false;
3  while converged = false do
4      converged ← true;
5      for i = 1 to |D| do
6          if y_i w · x_i ≤ 0      // i.e., ŷ_i ≠ y_i
7          then
8              w ← w + η y_i x_i;
9              converged ← false;  // We changed w so haven't converged yet
10         end
11     end
12 end
```

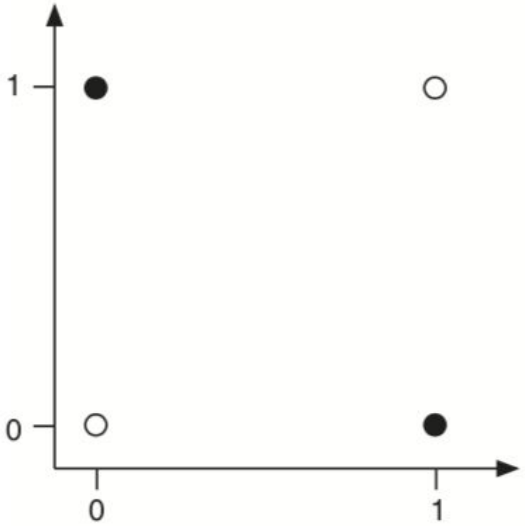
Multilayer perceptron

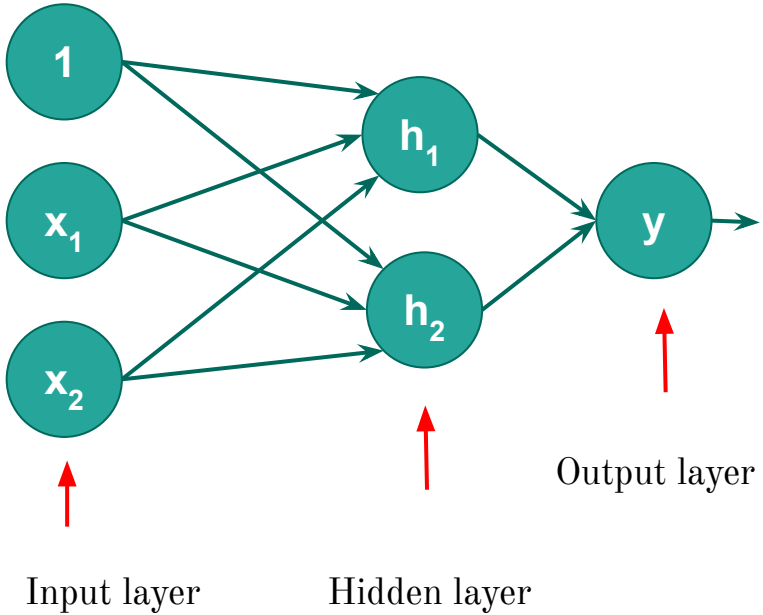
If the data are not linearly separable, a single perceptron is not sufficient, we need several of them.



Multilayer perceptron

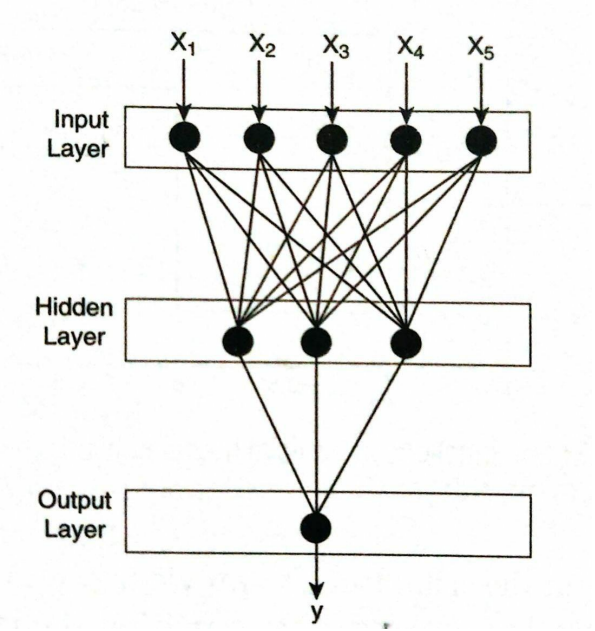
If the data are not linearly separable, a single perceptron is not sufficient, we need several of them.





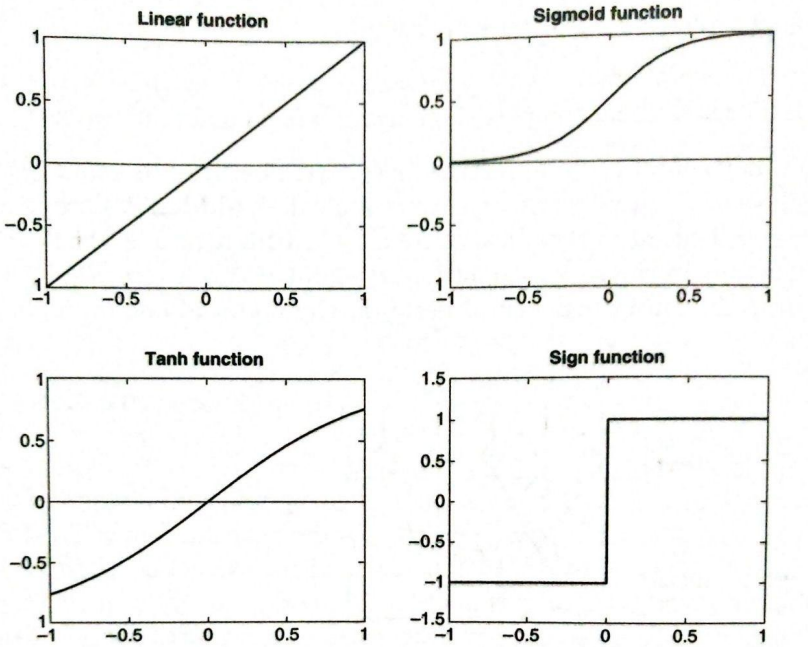
Multilayer Artificial Neural Network

- A multilayer Artificial Neural Network contains several intermediary layers (called **hidden layers**) between its input and output layers. Nodes in such layers are called **hidden nodes.**
- In a **feed-forward** neural network, the nodes in one layer are connected only to the nodes in the net layer.
- In a **recurrent** neural network, the links may connect nodes within the same layer or nodes from one layer to the previous layers.



Multilayer Artificial Neural Network

A multilayer Artificial Neural Network may use types of activation functions other than the sign function. Examples of activation functions: linear, sigmoid (logistic), hyperbolic tangent functions



Multilayer perceptron learning

Two aspects:

1. Learning the structure of the network
2. Learning the connection weights

Learning the ANN model

Given: A labeled data, and a fixed network structure

Task: Determine appropriate weights for the connections in the network

Idea:

For each instance in the training data, modify the weights so as to minimize the mean squared error between the network's prediction and the actual target value (using the method of gradient descent).

$$E(W) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Learning the ANN model

The weight update formula used by the gradient descent method can be written as

$$w_j = w_j - \lambda \frac{\partial E(\mathbf{w})}{\partial w_j}$$

where λ is the learning rate.

The second term states that the weight should be increased in a direction that reduces the overall error term.

Because the error function is nonlinear, it is possible that the gradient descent method may get trapped in a local minimum.

Backpropagation

Gradient descent with backpropagation is commonly used to train feed-forward neural networks.

There are two phases in each iteration of the algorithm:

1. Forward phase: The weights obtained from the previous iteration are used to compute the output value of each neuron in the network.
2. Backward phase: The weight update formula is applied in the reverse direction, i.e. the weights at level k+1 are updated before weights at level k are updated.

Support Vector Machine (SVM)

Consider the training dataset D with d attributes and 1 target-variable (real number)

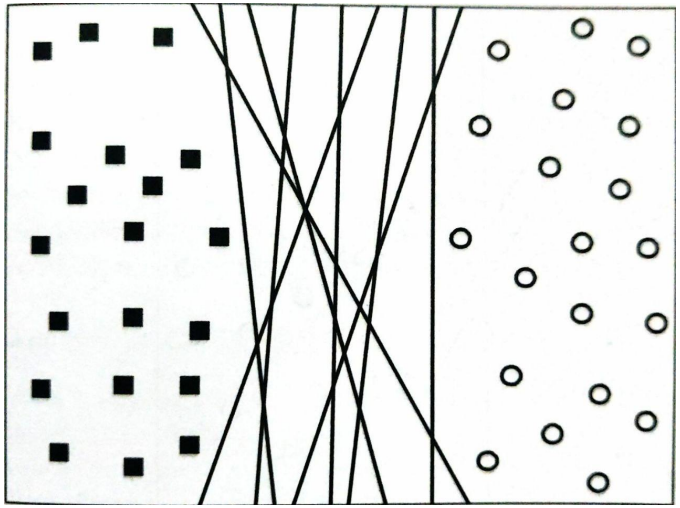
$$\{(X_i, y_i)|i = 1, 2, \ldots, n\}$$

Where each X_i corresponds to the set of attributes of the i^{th} observation and y_i is the class label, $y_i \in \{-1, +1\}$. If X_i is labeled with $+1$ it belongs to the positive class, else to the negative class(-1).

If the dataset is linearly separable, we can find a **hyperplane** such that all instances with positive class reside on one side of the hyperplane and all instances with negative class on the other side.

Support Vector Machine (SVM)

If the dataset is linearly separable, we can find a **hyperplane** such that all instances with positive class reside on one side of the hyperplane and all instances with negative class on the other side.



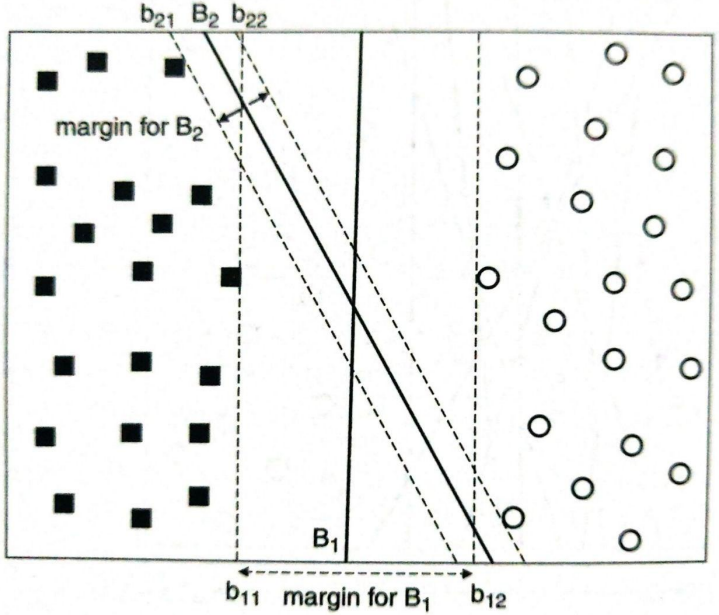
Support Vector Machine (SVM)

The decision boundary of a linear classifier can be written in the form:

$$wx - b = 0$$

Where w and b are parameters of the model.

SVM searches for the “best” hyperplane with the largest margin.

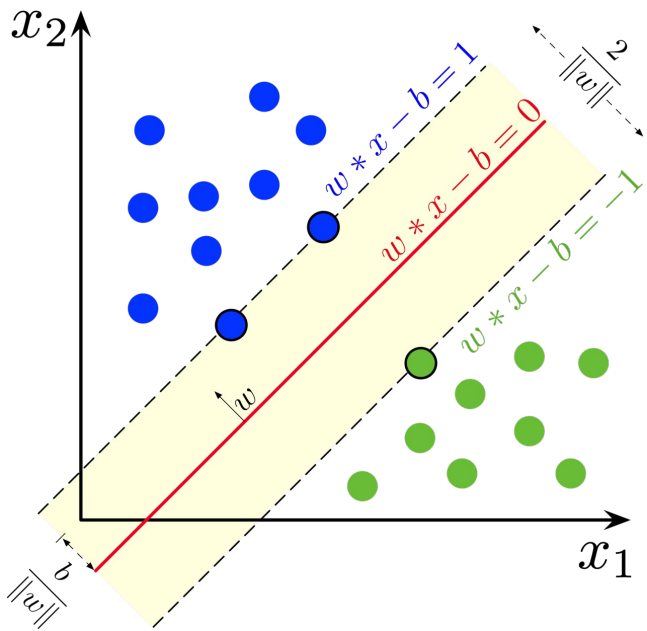


Support Vector Machine (SVM)

Consider X^+ and X^- the nearest positive and negative points for the hyperplane
$wx - b = 0$

We can rescale the parameters w and b of the decision boundary so that the two parallel hyperplanes passing through X+ and X- can be expressed as

H^+: $wx - b = 1$
H^-: $wx - b = -1$

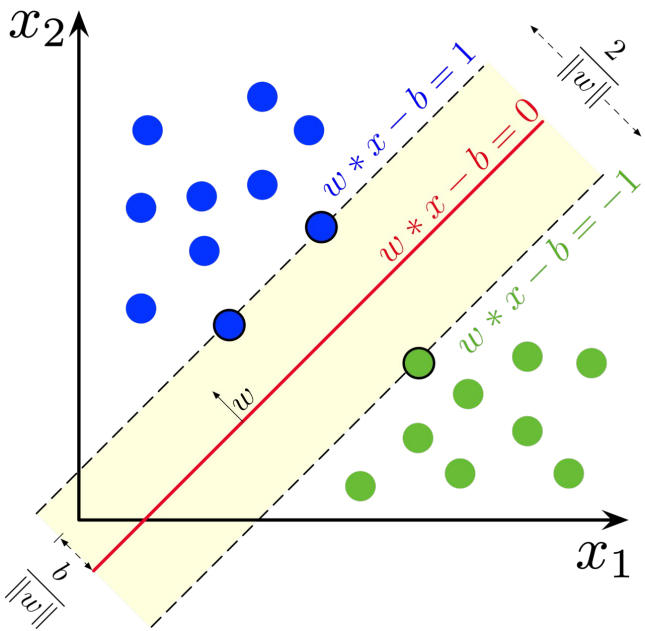


Support Vector Machine (SVM)

The margin is the distance between these two planes. It can be computed by substituting X^+ and X^- into the previous equations and subtracting the second equation from the first equation. Solving for margin, we get

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

The points X+and X-are called **support vectors.**



Support Vector Machine (SVM)

The training phase of SVM involves optimizing the following constrained optimization problem:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2$$

subject to $y_i(\mathbf{w}.\mathbf{x}_i - b) \geq 1, 1 \leq i \leq n$

References

1. https://computing.dcu.ie/~humphrys/Notes/Neural/single.neural.html
2. Bzdok, D., & Krzywinski, M., & Altman, N. (2018). Machine learning: Supervised methods. Nature Methods. 15. 10.1038/nmeth.4551
3. Cunningham, P., & Delany, S. J. (2020). k-Nearest Neighbour Classifiers 2nd Edition (with Python examples). arXiv preprint arXiv:2004.04523.