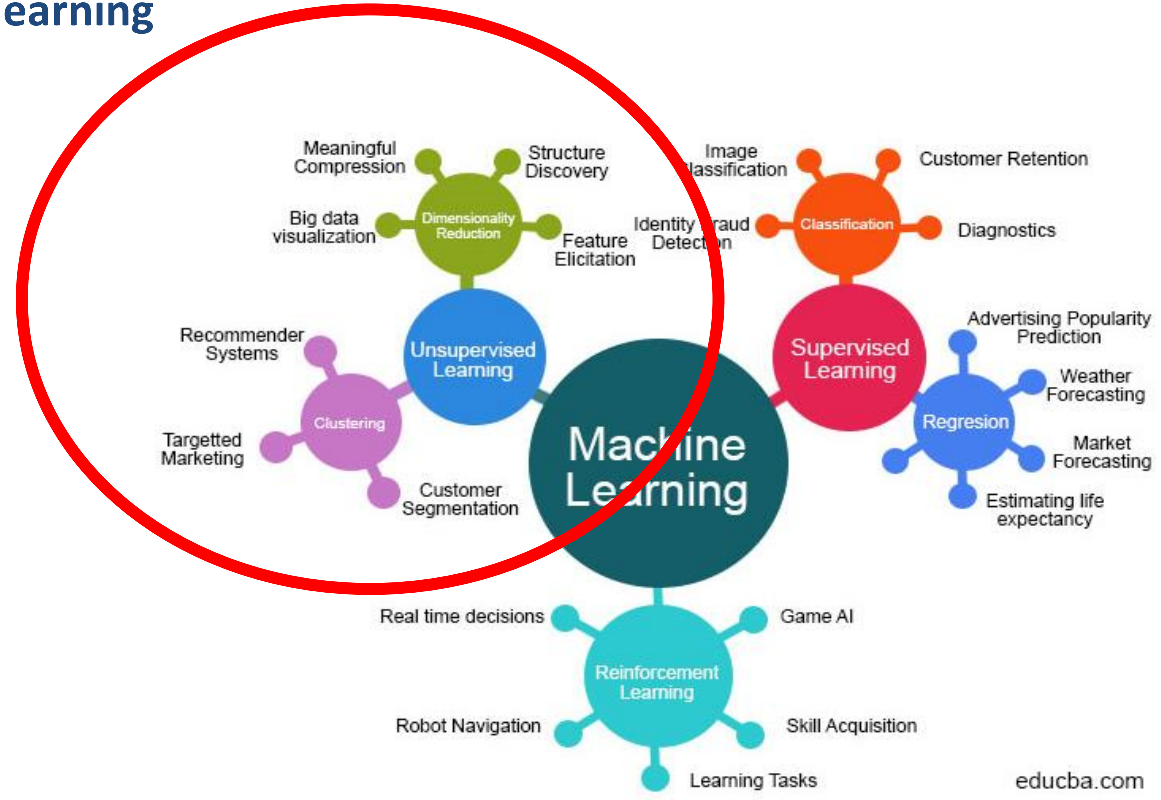
Models

Olivier Caelen

Machine learning

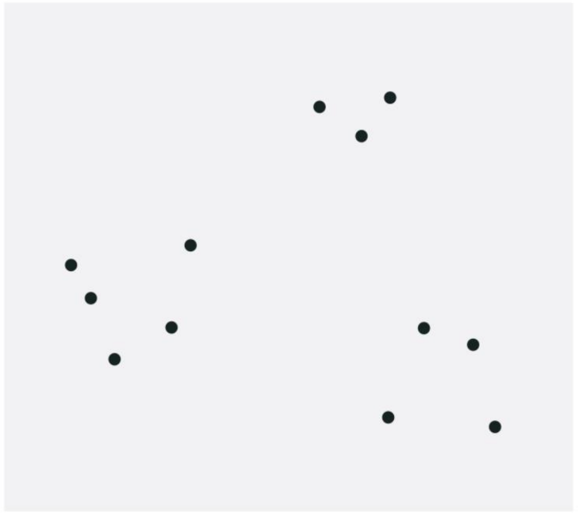


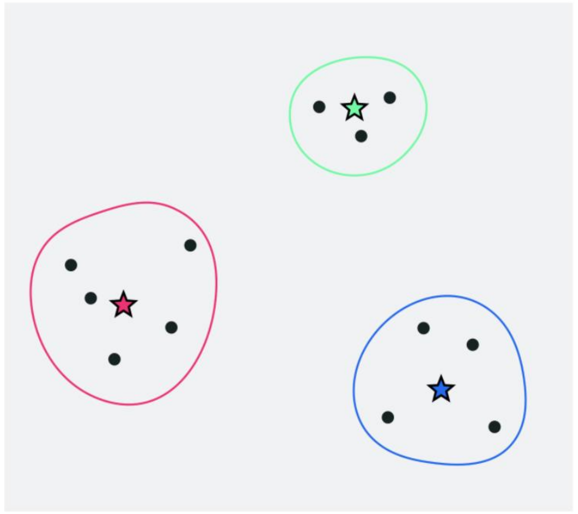
Unsupervised Learning

- Training data is a set of N input vectors x without any target labels.
  - Discovering groups of similar examples is called clustering, e.g. K-means clustering or Hierarchical clustering.
  - Determining the distribution of the data in the input space is called density estimation.
  - Projecting the data from a high-dimensional space is called visualization or dimensionality reduction, e.g. Principal Component Analysis (PCA).

K-means clustering

The goal is to discover K clusters, e.g. $K = 3$:





K-means clustering algorithm

Step by step:

- Choose K, the number of clusters
- Select K random centers μ_k (k = 1,… ,K)
- Assign each data point x_i to the “*closest*” centre, creating K clusters
- Update the center μ_k of each cluster
- Reassign each point x_i to the new closest center
- Repeat the 3 previous steps until convergence

Some questions/issues:

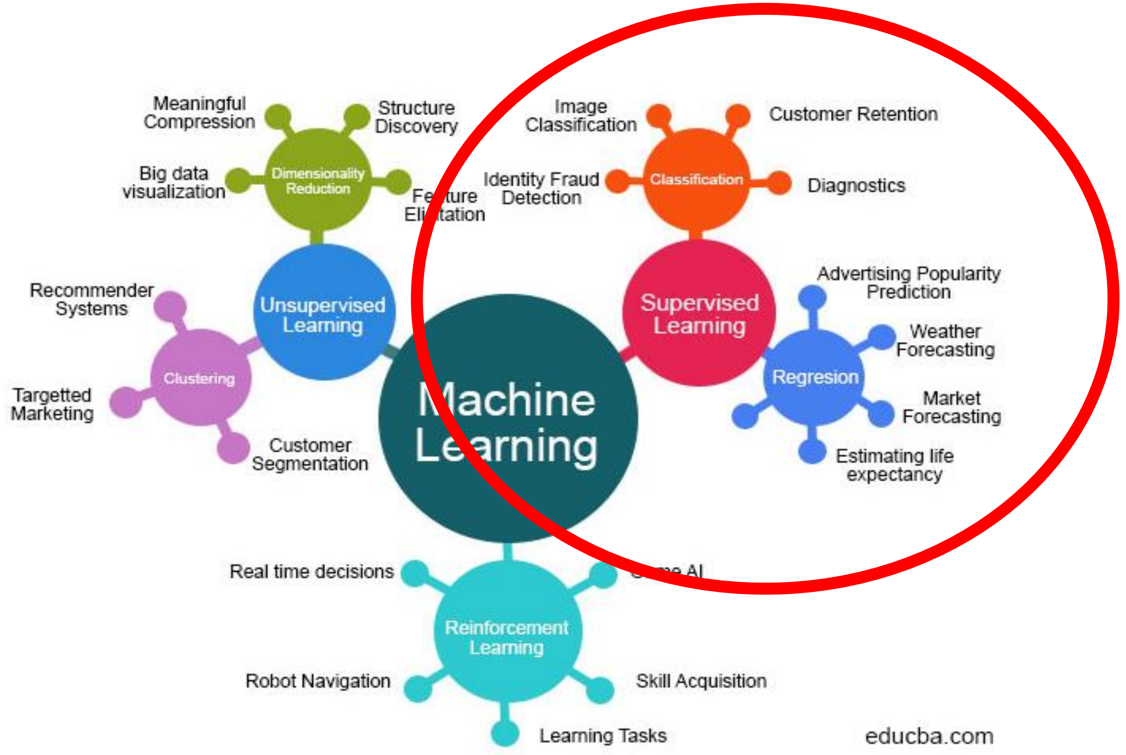
- Requires a distance or similarity measure!
- Importance of the choice of the initial centers? (empty clusters, sub-optimal clusters)
- Importance of the choice of K?
- Learning is minimizing the sum of squared distances of each point xi to its cluster centroid k :

$$\sum_{k=1}^{K} \sum_{x_i \; in \; cluster \; k} (x_i - \mu_k)^2$$

K-means example

https://www.youtube.com/watch?v=5I3Ei69I40s

Machine learning



Classification and Regression Trees

Trees - Concepts

Goal : predict an outcome based on a **set** of **rules**.

Trees are very transparent and easy to interpret.

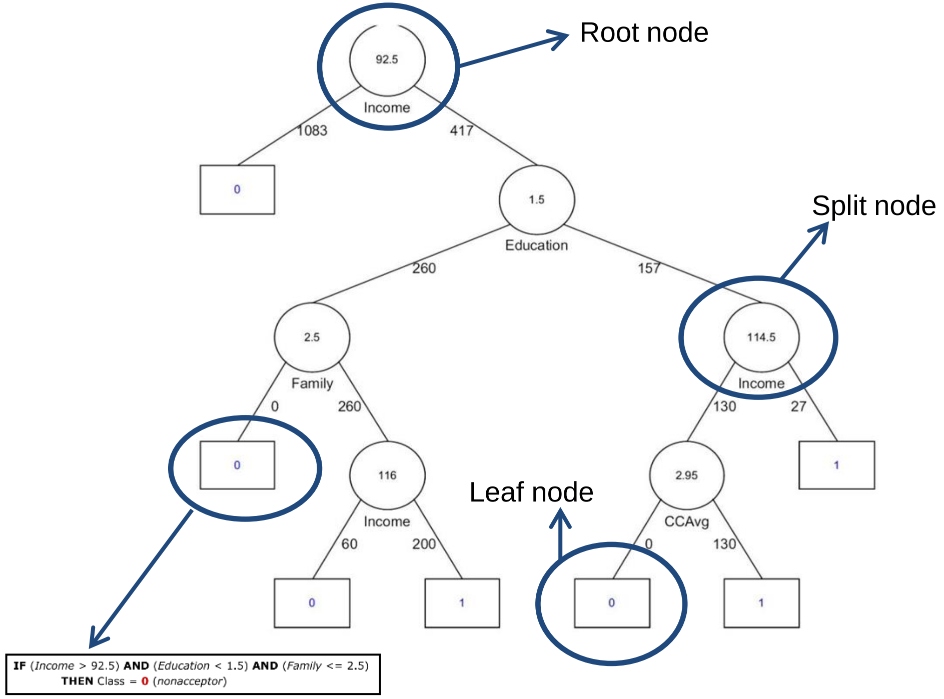
<u>Example</u>: classify a record as "*will accept credit card offer*" or "*will not accept*"

Income	Education	*Family*	Class
10,57	0,3	5,1	0
122,3	0,98	6,54	0
312,7	-1,3	4,4	1
10,8	0,01	8,5	0
70,5	1,1	1,4	1
31,2	1,2	2,4	1
…	…	…	…

A rule might be:

IF (*Income* > 92.5) **AND** (*Education* < 1.5) **AND** (*Family* <= 2.5)
THEN Class = **0** (*nonacceptor*)

Rules are represented by tree diagrams:



IF (*Income* > 92.5) **AND** (*Education* < 1.5) **AND** (*Family* <= 2.5)
THEN Class = **0** (*nonacceptor*)

Question for you: How many rules are in this tree?

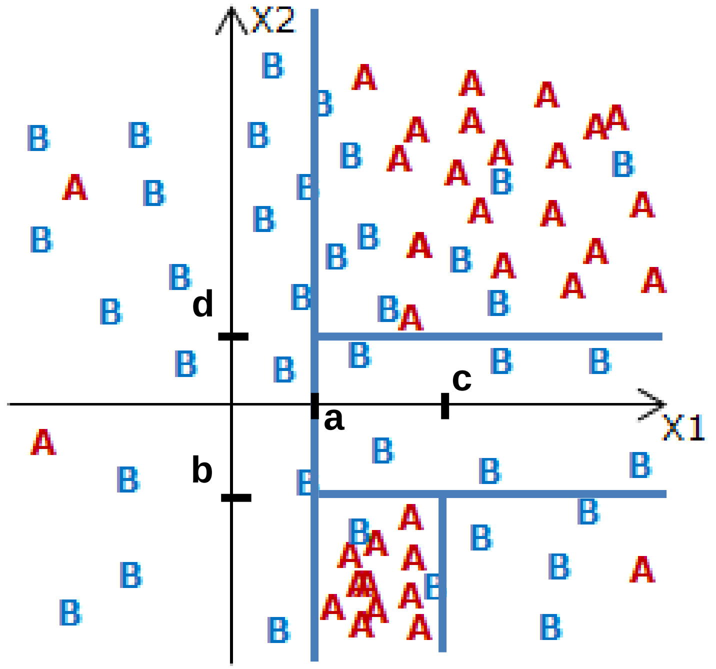
Trees – training Concepts

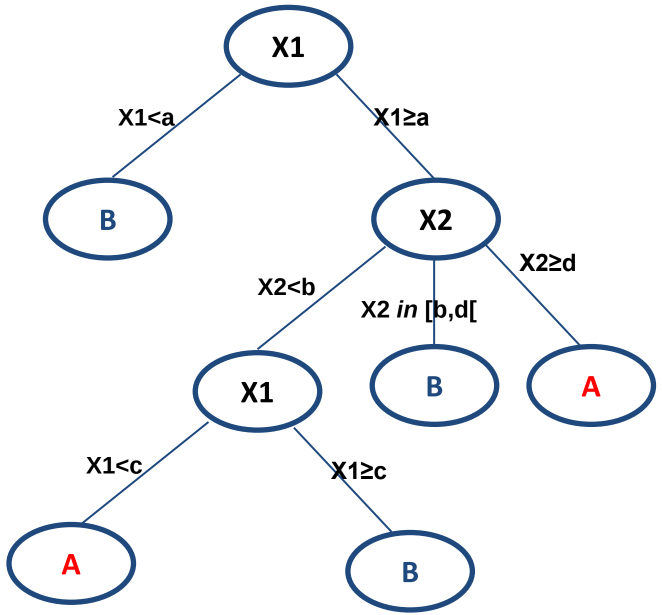
Key ideas:

- **Recursive partitioning:** Repeatedly split the records into two parts so as to achieve maximum homogeneity within the new parts.
  - We need a way to measure loss of homogeneity → Gini, Entropy, …
  - "Homogeneous" ≈ *containing records of mostly one class*

Question for you: And for regression?

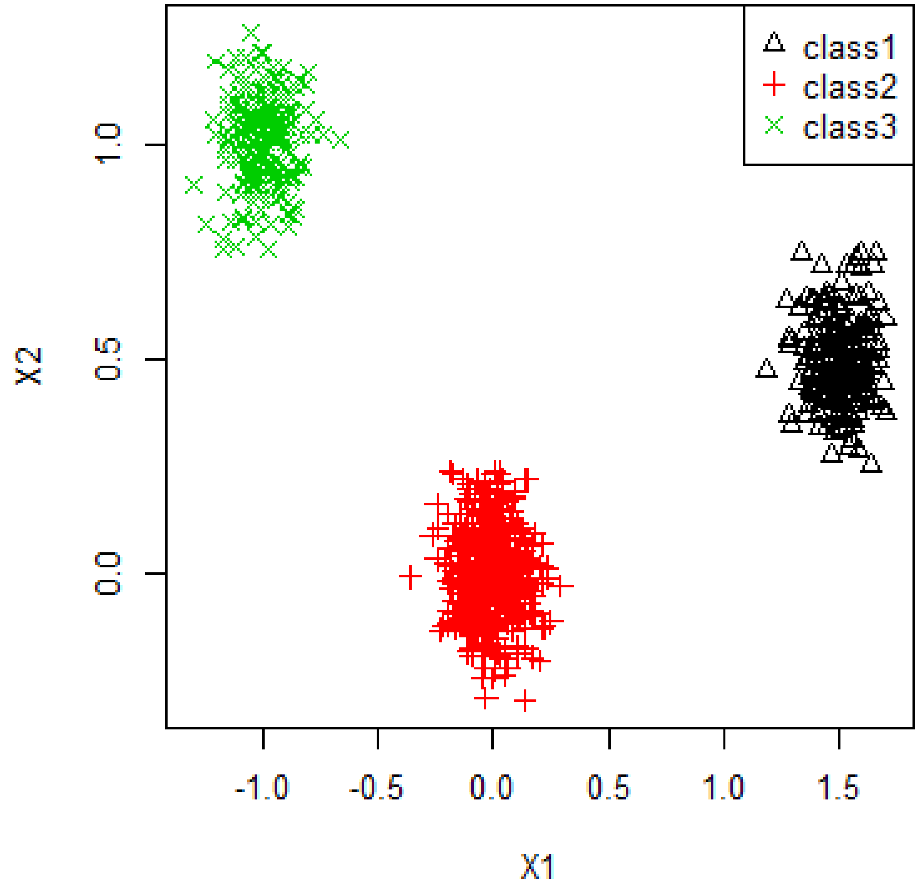
Recursive partitioning

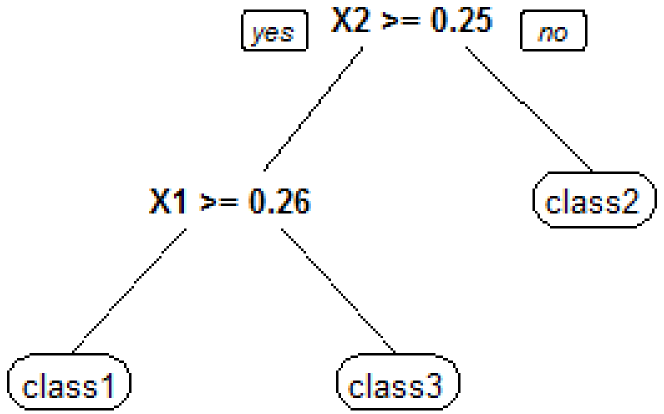




In this example, I did the partitioning manually.
We need a way to measure loss of homogeneity
→ Gini, Entropy, …

Question for you: Do you think that this tree is generated from this data set? (Yes/No)





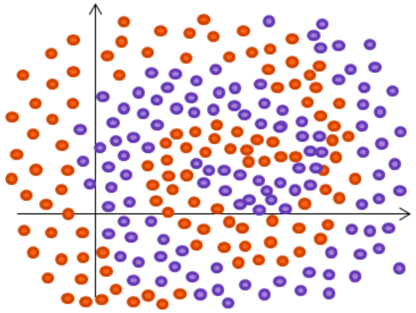
Recursive partitioning – algorithm

- Pick one of the predictor variables, $\boldsymbol{x}_i$
- Pick a value of $\boldsymbol{x}_i$, say $\boldsymbol{s}_i$, that divides the training data into two (not necessarily equal) portions
- Measure how homogeneous each of the resulting portions are

Algorithm tries different values of $\boldsymbol{x}_i$, and $\boldsymbol{s}_i$ to maximize purity in split

After you get a "maximum purity" split, repeat the process for a second split, and so on

Classification problems can be more challenging:



Think at this same problem in a higher dimension!

Measure of impurity

- There are several ways to measure impurity of a *Region*.

- The two most popular measures:
  - *Gini index*
  - *Entropy*.

- Let assume that we have two classes ($m = 2$).

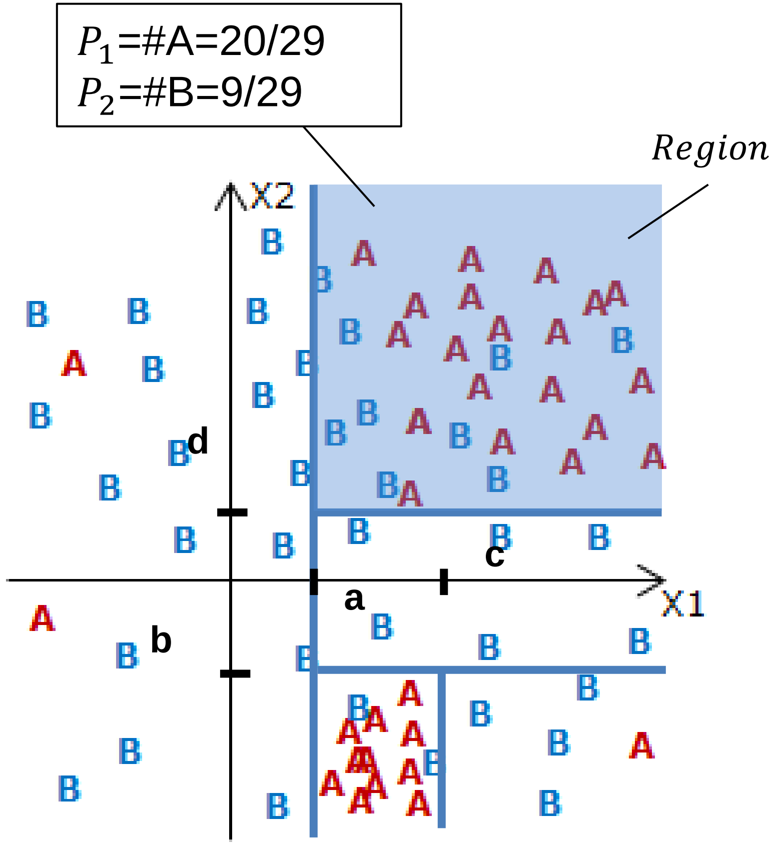
→ P_1=proportion of examples in first class

→ P_2=proportion of examples in second class

$$Gini(Region) = 1 - \sum_{k=1}^{m} p_k{}^2 = 1 - \left(\frac{20}{29}\right)^2 - \left(\frac{9}{29}\right)^2 \approx 0.428$$

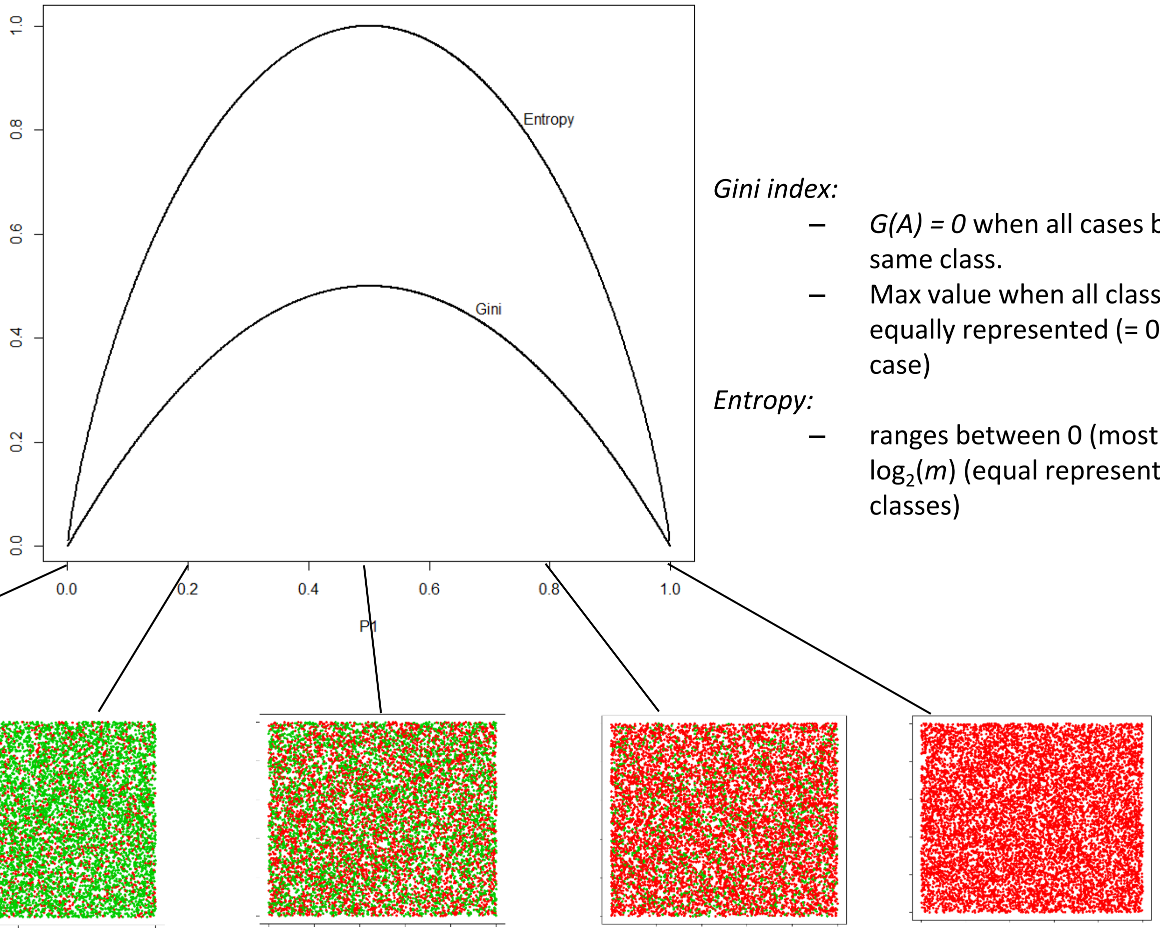
$$Entropy(Region) = -\sum_{k=1}^{m} p_k \log_2 p_k = -\left(\frac{20}{29}\right)\log_2\left(\frac{20}{29}\right) - \frac{9}{29}\log_2\left(\frac{9}{29}\right) \approx 0.894$$

Where m denote the number of classes.



Measure of impurity

Let assume that we have two classes.
P_1=proportion of examples in first class.



Gini index:

– *G(A) = 0* when all cases belong to same class.

– Max value when all classes are equally represented (= 0.50 in binary case)

Entropy:

– ranges between 0 (most pure) and $\log_2(m)$ (equal representation of classes)

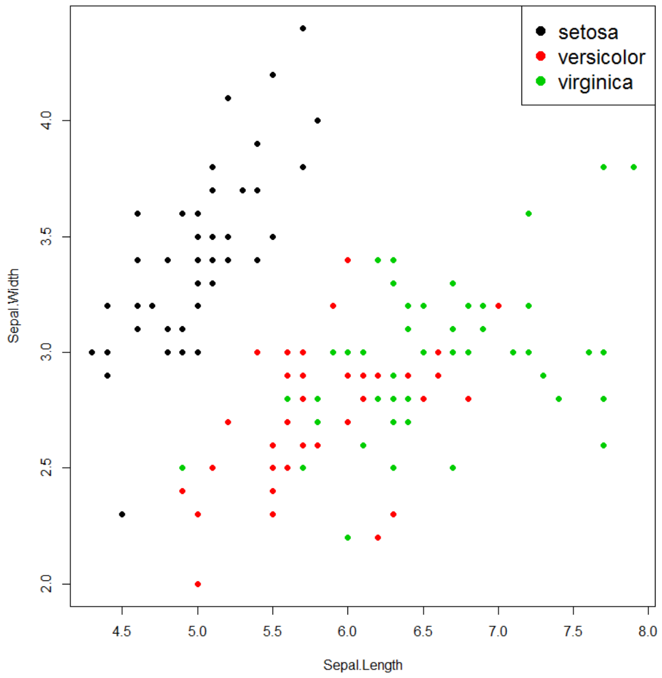
Impurity and Recursive Partitioning

- Obtain **overall** impurity measure (weighted avg. of individual rectangles)
- At each successive stage, compare this overall impurity measure across all possible splits in all variables
- Choose the split that reduces impurity the most
- Chosen split points become nodes on the tree
- Each leaf node label is determined by “voting” of the records within it, and by the cutoff value

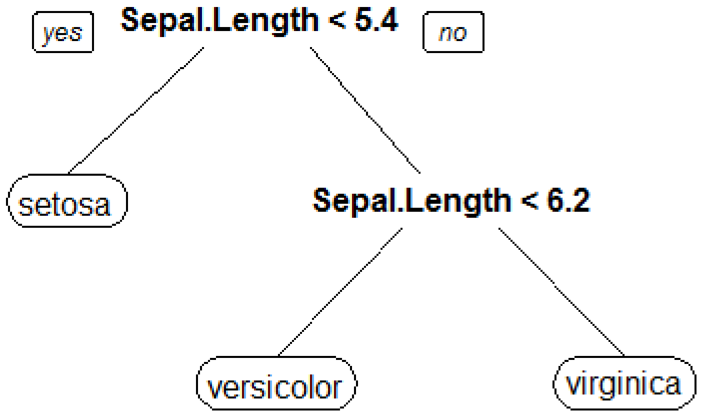
Tree Growth:

- Natural end of process is 100% purity in each leaf
- This **overfits** the data, <u>which end up fitting noise in the data</u>

Tree Growth

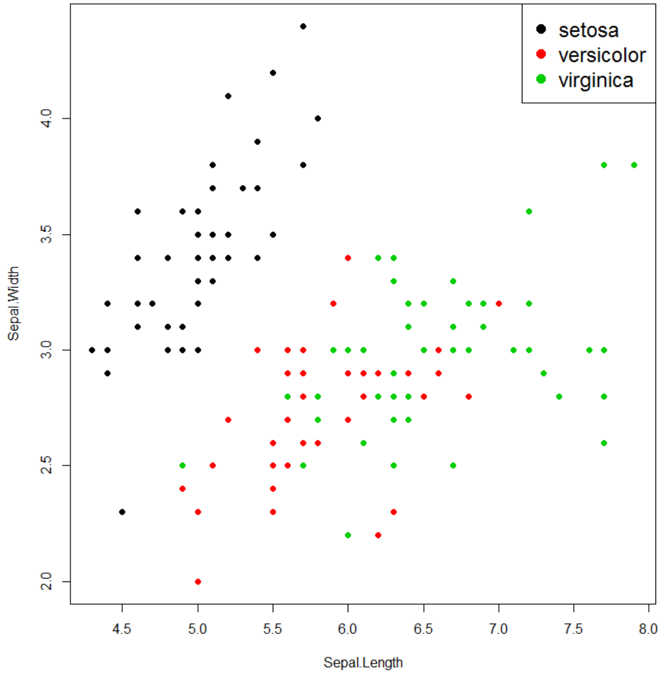


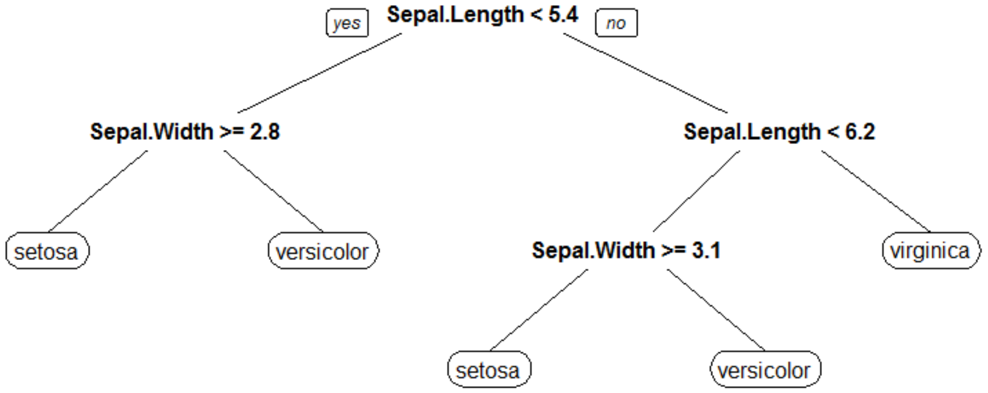
● setosa
● versicolor
● virginica
Sepal.Width
Sepal.Length
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



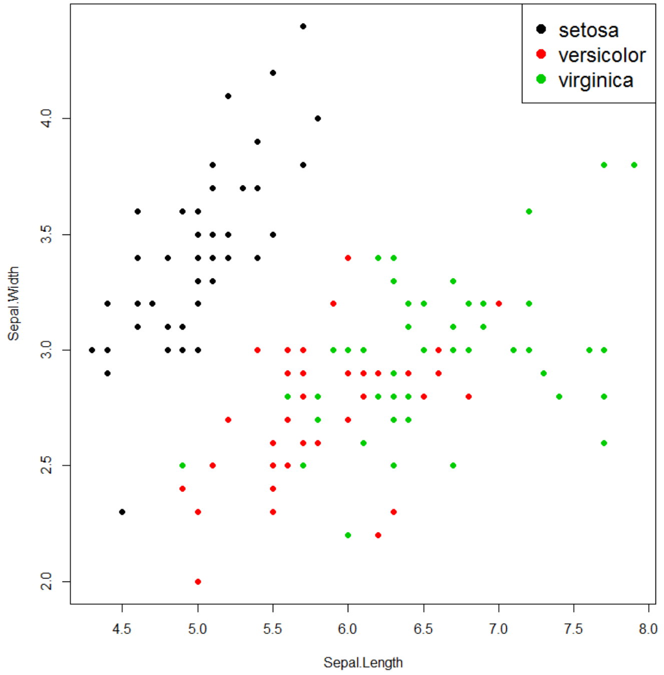
Sepal.Length < 5.4
yes
no
setosa
Sepal.Length < 6.2
versicolor
virginica

Tree Growth

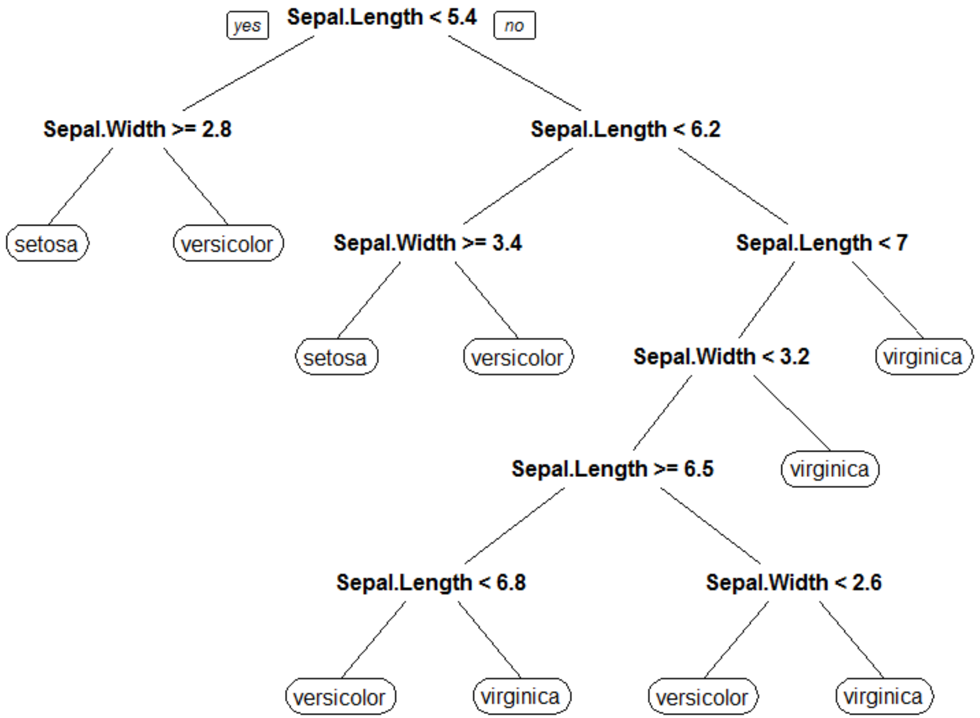

setosa
versicolor
virginica
Sepal.Width
Sepal.Length
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


yes
Sepal.Length < 5.4
no
Sepal.Width >= 2.8
Sepal.Length < 6.2
setosa
versicolor
Sepal.Width >= 3.1
virginica
setosa
versicolor

Tree Growth

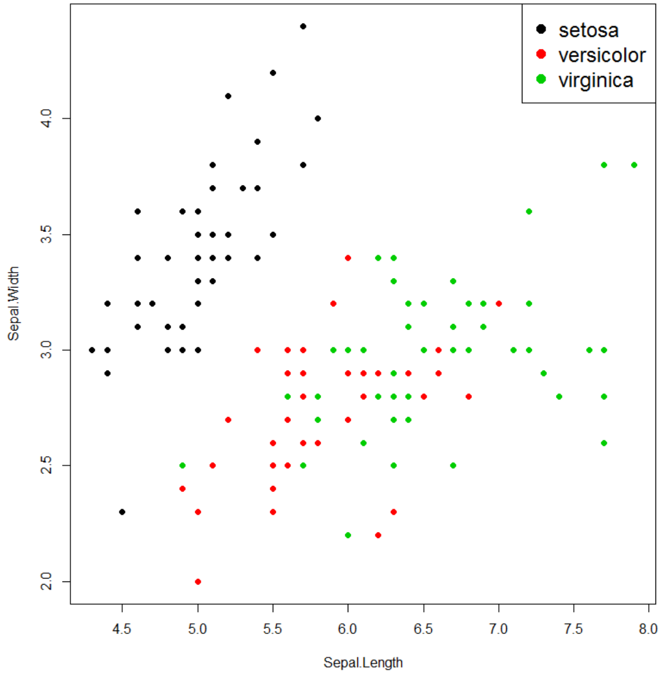


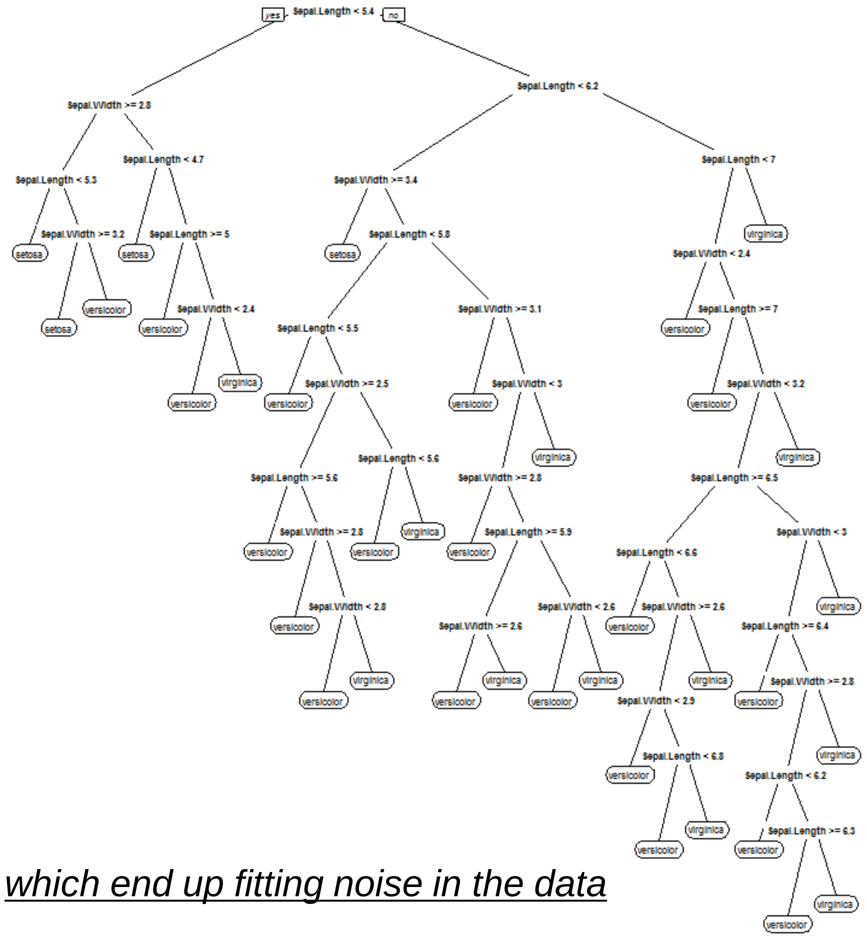
setosa
versicolor
virginica
Sepal.Width
Sepal.Length
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



Sepal.Length < 5.4
yes
no
Sepal.Width >= 2.8
Sepal.Length < 6.2
setosa
versicolor
Sepal.Width >= 3.4
Sepal.Length < 7
setosa
versicolor
Sepal.Width < 3.2
virginica
Sepal.Length >= 6.5
virginica
Sepal.Length < 6.8
Sepal.Width < 2.6
versicolor
virginica
versicolor
virginica

Tree Growth





This **overfits** the data, *which end up fitting noise in the data*

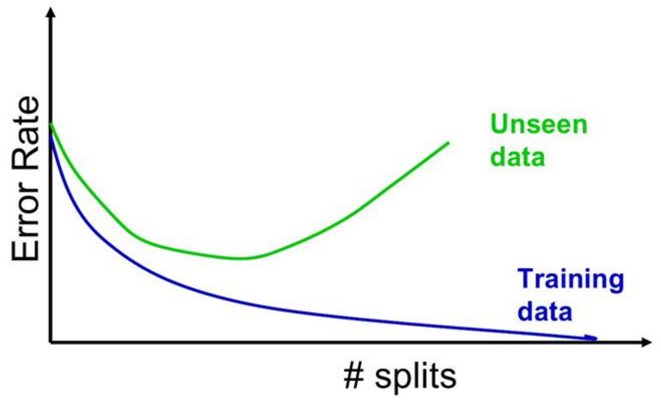
Tree Growth

Overfitting leads to low predictive accuracy of new data

Past a certain point, the error rate for the validation data starts to increase

Two main strategies to avoid Overfitting :

- Stopping Tree Growth
- Pruning



Regression Trees

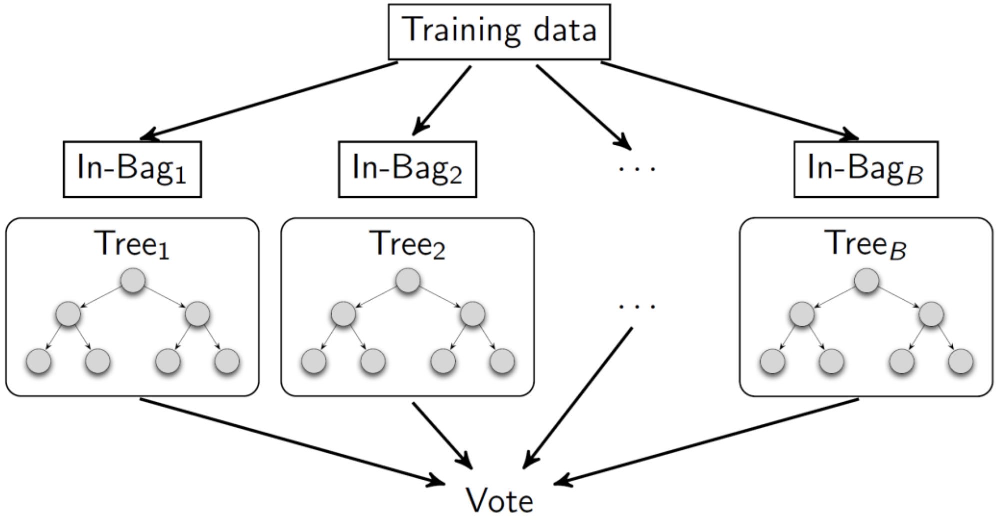
- Used with a continuous outcome variable
- Procedure similar to classification tree
  → Many splits attempted, choose the one that minimizes impurity
- Differences from classification tree
  - Prediction is computed as the **average** of numerical target variable in the rectangle (in classification tree it is majority vote)
  - Impurity measured by **sum of squared deviations** from leaf mean

X1	X2	Target
1,57	0,3	30
12,3	0,98	20
32,7	-1,3	30
1,8	0,01	40
7,5	1,1	12
3,2	1,2	15
1,2	-0,5	45
…	…	…

Random forest

Random forests

- Ensemble of decision trees, majority vote (classification) or average (regression).
- Training data is sampled with replacement (In-Bag).
- Only a small random subset of features is used at every split point



K Nearest Neighbors (KNN)

KNN

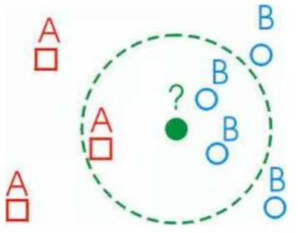
Different Learning Methods

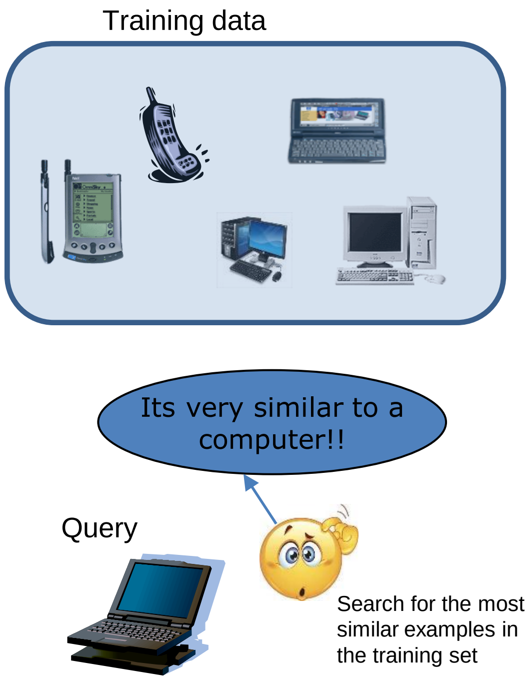
- Global Model
  - Explicit description of target function on the whole training set
- Instance-based Learning (local model)
  - Learning=storing all training instances
  - Prediction=assigning target function to a new instance
  - Referred to as "Lazy" learning

Lazy learning

- Does not "learn" until the test example is given
- Whenever we have a new data to classify, we find its K-nearest neighbors from the training data

- Classified by "MAJORITY VOTES" for its neighbor classes
  - Assigned to the most common class amongst its Knearest neighbors (by measuring "distant" between data)




Training data
Its very similar to a computer!!
Query
Search for the most similar examples in the training set

Distance measure

Calculate the distance between new example ('query') and all examples in the training set.

Distance measure for continues variables

- Euclidean

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Manhattan

$$\sum_{i=1}^{n}|x_i - y_i|$$

It is important to normalize or standardize in the inputs!

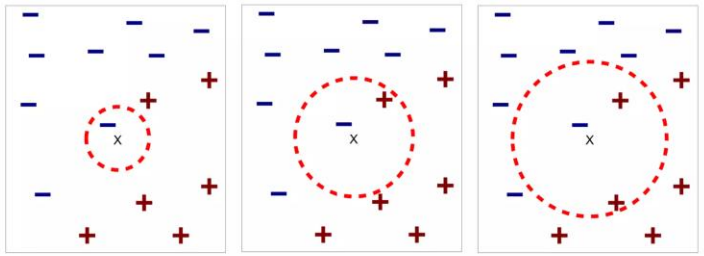
Normalize:

$$x^{new} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Standardize (z-score normalization):

$$x^{new} = \frac{x - \hat{\mu}}{\hat{\sigma}}$$

Question for you: why is it important to do this transformation?



(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.

Question for you:
What is the hyper parameter of the KNN method?
When does KNN over- and underfit?

Strengths of KNN

- **Very Simple and Intuitive**: The core idea behind KNN is based on the intuitive principle that similar things are near to each other. This makes the algorithm easy to understand, even for those who may not be experts in machine learning.
- **Easy to Update Training Data Set**: Since KNN is an instance-based learning algorithm, adding or removing data points can be done easily without requiring a full model retraining. This makes KNN well-suited for applications where the data set is evolving over time.
- **Adaptability**: KNN can be used for both classification and regression tasks. It's also versatile in handling multi-class problems. The algorithm can be easily adapted to various distance metrics like Euclidean, Manhattan, or more domain-specific measures.
- **Interpretable Results**: Although not as interpretable as decision trees or linear models, KNN's rationale ("it's similar to these known cases") is relatively easy to understand. This can be particularly useful when you need to explain the model's decisions.

Weaknesses of KNN

- **Computational Complexity**: KNN can be computationally expensive, especially with large datasets, as it needs to compute the distance to every point in the dataset for each prediction.
- **Memory Intensive**: The algorithm stores the entire dataset, making it memory-intensive. This can be a significant drawback when dealing with large datasets.
- **Curse of Dimensionality**: The algorithm suffers from the "curse of dimensionality." As the number of dimensions increases, the distance between points becomes less meaningful, which degrades the performance.
- **Parameter Sensitivity**: The choice of the number of neighbors (k) and the type of distance metric can greatly affect the performance, making it sometimes difficult to choose the optimal parameters.

High-dimensional data exist! *Curse of dimensionality*

Low dimension (2D)

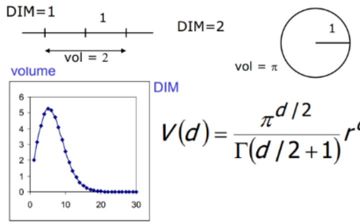
- Situation we can imagine, represent, draw, …
- Strong intuition of how the tools behave
- Consider cases where :
  #observation >> #dimension

High dimension

- No representation
- No intuition
- Sometime: #observation << dimension

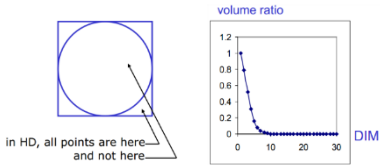
No good intuition in high dimension

Volume of a sphere of constant radius (=1) in dimension DIM

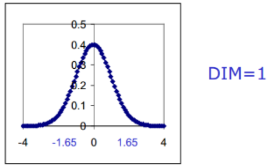


$$V(d) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d$$

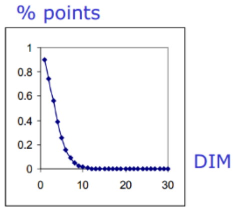
Ratio volume shere / cube



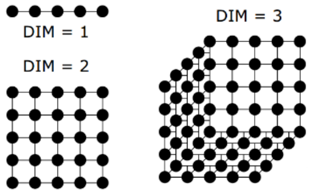
Muli-DIM Gaussian distribution



% points inside a sphere of radius 1.65:



To keep the same density of points, the number of points on a grid increases exponentially with DIM!



Naive Bayes classifier

*“**Naive Bayes classifiers** are a family of simple probabilistic classifiers based on applying **Bayes' theorem** with strong **independence assumptions** between the features.”*

WIKIPEDIA
The Free Encyclopedia

$\boxed{\text{Naive}}\,\boxed{\text{Bayes}}$ classifier

Conditional probability distribution (In the discrete case):

- Let X and Y be two discrete random variables.
- Let $P(X = x)$ and $P(Y = y)$ be the corresponding mass functions.
- The conditional probability mass function of Y is defined as follow:

$$\boldsymbol{P(Y = y \mid X = x) = \frac{P(X = x \,\cap\, Y = y)}{P(X = x)}}$$

Assuming that the event $X = x$ is realized, $P(Y = y \mid X = x)$ is the probability that the event $Y = y$ will happen.

- Let X and Y be two continues random variables.
- Let $f_X(x)$ and $f_Y(y)$ be the corresponding density functions.
- The conditional density function of Y is defined as follow:

$$\boldsymbol{f_{Y|X}(y|x) = \frac{f_{XY}(x, y)}{f_X(x)}}$$

Independence

Two (discrete/continues) random variables X and Y are independent if:

$$\forall\, \boldsymbol{x} \in \mathbf{supp}(\boldsymbol{X}), \forall\, \boldsymbol{y} \in \mathbf{supp}(\boldsymbol{Y}):$$

$$\boldsymbol{P}(\boldsymbol{X} = \boldsymbol{x} \cap \boldsymbol{Y} = \boldsymbol{y}) = \boldsymbol{P}(\boldsymbol{X} = \boldsymbol{x})\, \boldsymbol{P}(\boldsymbol{Y} = \boldsymbol{y})$$ *discrete case*

$$\boldsymbol{f}_{\boldsymbol{XY}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{f}_{\boldsymbol{X}}(\boldsymbol{x})\, \boldsymbol{f}_{\boldsymbol{Y}}(\boldsymbol{y})$$ *continues case*

Notation:
"X and Y are independent" $\Leftrightarrow$ $X \perp Y$

Some properties

$$X \perp Y \quad \Leftrightarrow \quad P(Y = y \mid X = x) = P(Y = y)$$

The probability of $Y = y$ do not depend on the value taken by the random variable X.

$$X \perp Y \quad \Rightarrow \quad E[XY] = E[X]\, E[Y]$$

Independence is a stronger assumption than uncorrelated!

$$\boxed{X \perp Y \quad \Rightarrow \quad \mathrm{cov}(X, Y) = 0}$$

- If X and Y are independent then X and Y are uncorrelated.
- But the opposite is not always true.

Bayes' theorem

$$P(Y=y \mid X=x) = \frac{P(X=x \cap Y=y)}{P(X=x)} \qquad P(X=x \mid Y=y) = \frac{P(X=x \cap Y=y)}{P(Y=y)}$$

$$\boxed{P(Y=y \mid X=x) = \frac{P(X=x \mid Y=y)\; P(Y=y)}{P(X=x)}}$$

If Y is the *output* and X is the *input*

- $P(Y=y)$: prior probability of the *output*
- $P(Y=y \mid X=x)$: posterior probability of the *output* given the *input* x
- $P(X=x \mid Y=y)$: posterior probability of the *input* given the *output* y

Maximum a posteriori methods (MAP) for machine learning

$$\hat{y} = \underset{y \in \{C_1, \cdots, C_k\}}{\operatorname{argmax}} \; P\,(Y = y \mid X = x)$$

$$= \underset{y \in \{C_1, \cdots, C_k\}}{\operatorname{argmax}} \; \frac{\boldsymbol{P(X = x \mid Y = y)\; P(Y = y)}}{\boldsymbol{P(X = x)}}$$

$$= \underset{y \in \{C_1, \cdots, C_k\}}{\operatorname{argmax}} \; P\,(X = x \mid Y = y)\; P(Y = y)$$

MAP classifier requires reliable estimates for $P(X = x \mid Y = y)$ and $P(Y = y)$.

Question for you: Given the following dataset, what is the estimation of $P(Y = y)$?

X1	X2	Y
12	12	C_2
5	13	C_1
45	5	C_1
24	541	C_2
5	15	C_1
8	52	C_3
7	12	C_1

$\hat{P}(Y = C_1) =?$

$\hat{P}(Y = C_2) =?$

$\hat{P}(Y = C_3) =?$

We still have to estimate $P(X = x \mid Y = y)$!

How to estimate $P(X = x \mid Y = y)$?

- Assume the input space X is characterized by $d = 20$ features having 10 possible values. The data likelihood is defined by a joint conditional probability:
$$P(x \mid y) = P(\{X_1 = x_1, \cdots, X_{20} = x_{20}\} \mid Y = y)$$
- If we have no other information, this requires the estimation of 10^{20} values (!!) for each classes.
- One solution: assume conditional independence between the features (Naïve Bayes classifier).

Naive Bayes classifier

Discrete input variables

- Assume target function $f: X \rightarrow Y$, where each example x is described by d features $\{x_1, x_2, \cdots, x_d\}$.

- For each class y, we need to estimate

$$P(x \mid y) = P(\{X_1 = x_1, \cdots, X_d = x_d\} \mid Y = y)$$

- **Naive Bayes assumption**:

$$P(\{X_1 = x_1, \cdots, X_d = x_d\} \mid Y = y) = \prod_{j=1}^{d} P\left(X_j = x_j \,\middle|\, Y = y\right)$$

$\boxed{\text{Assume } \underline{\text{conditional}} \text{ independence}}$

- **Naive Bayes classifier:**

$$\boldsymbol{\hat{y} = \underset{y \in \{C_1, \cdots, C_k\}}{\mathrm{argmax}}\, \hat{P}(Y = y) \prod_{j=1}^{d} \hat{P}\left(X_j = x_j \,\middle|\, Y = y\right)}$$

Question for you

Given the following dataset,

X1	X2	Y
A	α	C_2
A	β	C_1
B	β	C_1
C	δ	C_2
C	α	C_1
A	α	C_2
B	β	C_1

What is $\hat{P}(X1 = A\,,\ X2 = \alpha \mid Y = C_2)$ = ? (With conditional independence assumption)

Solution: $\frac{2}{3} = 0.6666\ldots$

What is $\hat{P}(X1 = A\,,\ X2 = \alpha \mid Y = C_2)$ = ? (Without conditional independence assumption)

Solution: $\frac{4}{9} = 0.4444\ldots$

Naive Bayes classifier – example

Discrete input variables

$$\hat{y} = \underset{y \in \{C_1, \cdots, C_k\}}{\text{argmax}} \; P\,(X = x|\; Y = y)\; P(Y = y)$$

Given the following dataset, we will use a Naive Bayes classifier to estimate $\hat{y}$ when $X1 = A\,,\; X2 = \alpha$.

X1	X2	Y
A	α	C_2
A	β	C_1
B	β	C_1
C	δ	C_2
C	α	C_1
A	α	C_2
B	β	C_1

- **Y=$\boldsymbol{C_2}$**

$\hat{P}(Y = C_2) = {}^3/_7$

$\hat{P}(X1 = A\,, X2 = \alpha \,|\, Y = C_2) = {}^4/_9$ (assuming $X_1 \perp X_2 \,|\, Y = C_2$)

$\hat{P}(X1 = A\,, X2 = \alpha \,|\, Y = C_2) \;\times\; \hat{P}(Y = C_2) \;=\; \frac{3}{7} \times \frac{4}{9} \approx\; 0.1905$

- **Y=$\boldsymbol{C_1}$**

$\hat{P}(Y = C_1) = {}^4/_7$

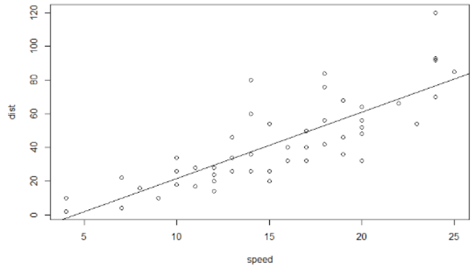
$\hat{P}(X1 = A\,, X2 = \alpha \,|\, Y = C_1) = {}^1/_4 \times {}^1/_4 = {}^1/_8$

$\hat{P}(X1 = A\,, X2 = \alpha \,|\, Y = C_1) \;\times\; \hat{P}(Y = C_1) \;=\; \frac{4}{7} \times \frac{1}{8} \approx\; 0.0714$

$$\boldsymbol{\hat{y}} = \underset{y \in \{C_1, \cdots, C_k\}}{\text{argmax}} \; P\,(X = x|\; Y = y)\; P(Y = y) \;=\; \boldsymbol{C_2}$$

Linear models

Linear regression



Question for you: What is the prediction of dist when speed = 7?

Here, we assume a linear dependency between input and output.

Set of all the linear models with two parameters and only one input/output:

$$\Lambda = \{y = w_1.x + w_0 \mid w_1 \in \mathbb{R}, w_0 \in \mathbb{R}\}$$

If Λ contains linear models, then ‘Least squares’ is a well-known parametric identification algorithm.

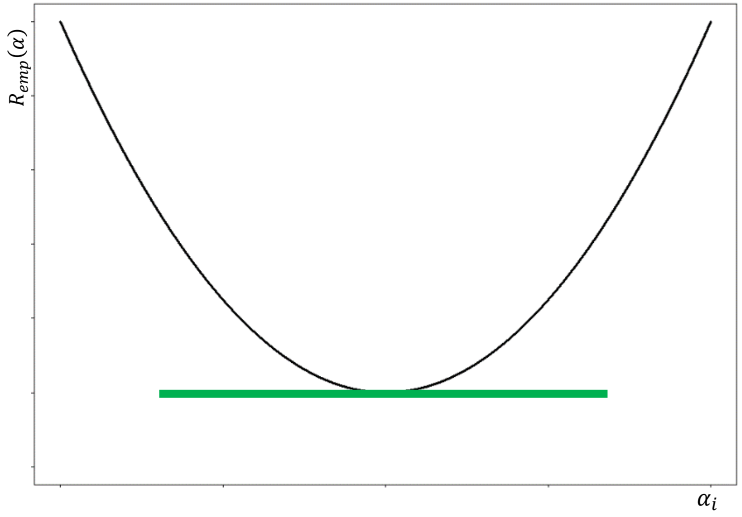
The empirical risk function (aka: error on the training set):

$$R_{emp}(\alpha) = \sum_{i=1}^{N} (y_i - h(x_i, \alpha))^2$$
$$= \sum_{i=1}^{N} (y_i - w_0 - w_1.x_i)^2$$

We will see that $R_{emp}(\alpha)$ is a convex function for $\alpha_0$or α_1.

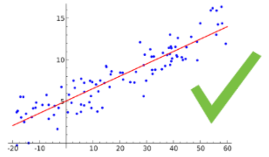
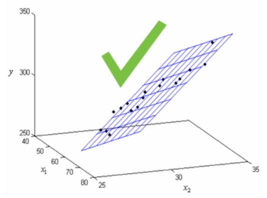
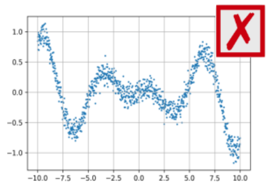
And as the function is convex, the minimum of $R_{emp}(\alpha)$ is the value for which the slope of tangent line of $R_{emp}(\alpha)$ equals zero.

In machine learning, it is common to rename $R_{emp}(\alpha)$ by $J(\alpha)$.



Linear regression

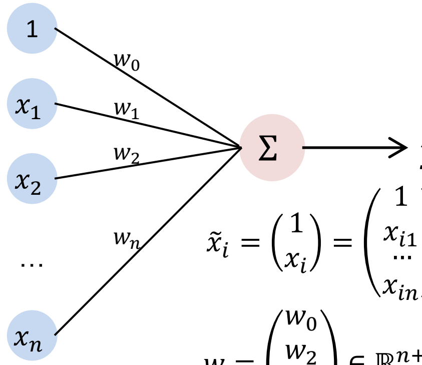
- Probably the most elementary way to perform regression.
- It is a linear model (cannot capture nonlinear relations)

- Let $R_{emp}(\alpha)$ be the loss function that we can define in regression as follow

$$R_{emp}(\alpha) = \sum_{i=1}^{N}(h(x_i) - y_i)^2 = \sum_{i=1}^{N}(w^T\tilde{x}_i - y_i)^2 = \left\|\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right\|^2$$

where $\tilde{\boldsymbol{X}} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1n} \\ 1 & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nn} \end{pmatrix} \in \mathbb{R}^{N\times(n+1)}$ and $\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$



$$\tilde{x}_i = \begin{pmatrix} 1 \\ x_i \end{pmatrix} = \begin{pmatrix} 1 \\ x_{i1} \\ \cdots \\ x_{in} \end{pmatrix} \in \mathbb{R}^{n+1}$$

$$w = \begin{pmatrix} w_0 \\ w_2 \\ \cdots \\ w_n \end{pmatrix} \in \mathbb{R}^{n+1}$$

$$h(x_i) = w^T\tilde{x}_i = w_0 + \sum_{j=1}^{n} x_{ij}w_j$$

where "h"stands for *hypothesis*

<u>Learning</u>: Given a training set d, search the optimal parameters w minimizing $R_{emp}(\alpha)$ where $\alpha = \mathrm{w}$.

Parametric identification in linear regression context

Parametric identification : Given a training set d, search the optimal parameters w minimizing R_{emp}.

- Learning is an optimization problem.
  - Therefore, many optimization algorithms can be used to find the best w.
- We will search analytically the parameters for which the gradient of R_{emp} is zero.
  - i.e., $\left(\frac{\partial R_{emp}}{\partial w}\right) = 0$
  - Note, a direct analytical solution is possible due to the fact when a linear model is used (i.e., $h(\mathrm{x_i}) = w^T\tilde{x}_i$) then the second derivatives $\left(\frac{\partial^2 R_{emp}}{\partial (w_0)^2}\right)$ and $\left(\frac{\partial^2 R_{emp}}{\partial (w_1)^2}\right)$ are always positive

    → therefore, R_{emp} is a convex function → only one minimum

Let us consider the case with only one input and one output variables.
A function is convex in $\mathbb{R}$ if the second derivative is always positive.

$$R_{emp} = \sum_{i=1}^{N}(y_i - w_0 - w_1.x_i)^2$$

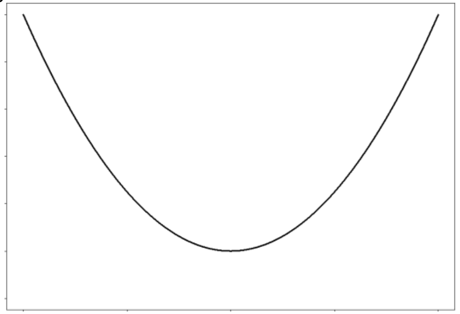
$$\frac{\partial R_{emp}}{\partial w_0} = -2\sum_{i=1}^{N}(y_i - w_0 - w_1.x_i)$$
$$= -2\sum_{i=1}^{N} y_i + 2Nw_0 + 2w_1\sum_{i=1}^{N} x_i$$

$$\frac{\partial^2 R_{emp}}{\partial {w_0}^2} = 2N > 0$$

$$\frac{\partial R_{emp}}{\partial w_1} = -2\sum_{i=1}^{N} x_i(y_i - w_0 - w_1.x_i)$$
$$= -2\sum_{i=1}^{N} x_i y_i + 2w_0\sum_{i=1}^{N} x_i + 2w_1\sum_{i=1}^{N} {x_i}^2$$

$$\frac{\partial^2 R_{emp}}{\partial w_1^2} = 2\sum_{i=1}^{N} {x_i}^2 > 0$$

We know that the function is convex.
→ Only one global minimum.



Optimizing the criterion by pseudo-inverse

- Error criterion $\mathrm{R}_{emp} = \frac{1}{2}\left\|\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right\|^2$(Note, for simplicity I add $^1/_2$)
- Gradient of R_{emp} (function to minimize) with respect to weights (free parameters).

$$\nabla_w(R_{emp}) = \left(\frac{\partial \mathrm{R}_{emp}}{\partial w}\right) = \left(\frac{\partial \mathrm{R}_{emp}}{\partial w_0}, \frac{\partial R_{emp}}{\partial w_2}, \dots, \frac{\partial R_{emp}}{\partial w_n}\right)^T \in \mathbb{R}^{n+1}$$

$$\frac{\partial \mathrm{R}_{emp}}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\frac{1}{2}\left\|\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right\|^2\right) = \frac{2}{2}\left(\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right)^T\left(\frac{\partial}{\partial w_j}\tilde{\boldsymbol{X}}w\right) = \left(\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right)^T x_{.j}$$

$$\nabla_w(R_{emp}) = \left(\left(\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right)^T\tilde{\boldsymbol{X}}\right)^T = \tilde{\boldsymbol{X}}^T\left(\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right)$$

Minimum of error

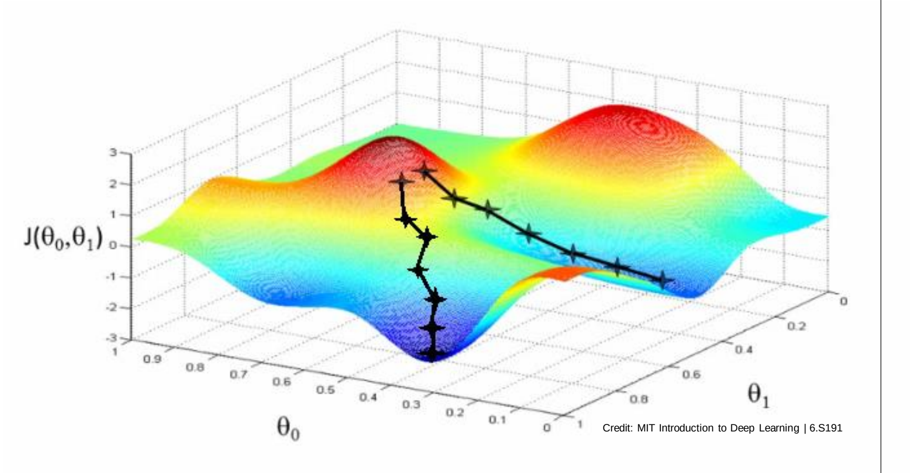
$$\left(\frac{\partial J}{\partial w}\right) = \mathbf{0}$$
$$\Leftrightarrow \tilde{\boldsymbol{X}}^T\left(\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right) = \mathbf{0}$$
$$\Leftrightarrow \tilde{\boldsymbol{X}}w = \boldsymbol{y}$$
$$\Leftrightarrow \tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}w = \tilde{\boldsymbol{X}}^T\boldsymbol{y}$$
$$\Leftrightarrow \left(\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}w = \left(\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^T\boldsymbol{y}$$
$$\Rightarrow \hat{w} = \left(\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^T\boldsymbol{y}$$

Pseudo-inverse requires:
- All input-output pairs from D_N
- matrix inversion (often ill-configured) (why?)

Necessity for iterative methods without matrix inversion
→ Gradient descent !

Training: Batch gradient descent (BGD)

General form: $w(t+1) = w(t) - \alpha \frac{\partial J}{\partial w}|_{w(t)}$



- The value of the gradient $\nabla_w R_{emp}$ at $w(t)$ gives the *local* direction of the steepest *increasing* slope.
- At each step t, we move in the *opposite* direction of the gradient.
- The parameter α (*learning rate*) is used to adapt the step sizes when moving.

When error criterion R_{emp} equals $\frac{1}{2}\left\|\tilde{\boldsymbol{X}}w - \boldsymbol{y}\right\|^2$

$$w(t+1) = w(t) + \alpha\, \tilde{\boldsymbol{X}}^T\left(\boldsymbol{t} - \tilde{\boldsymbol{X}}w(t)\right)$$

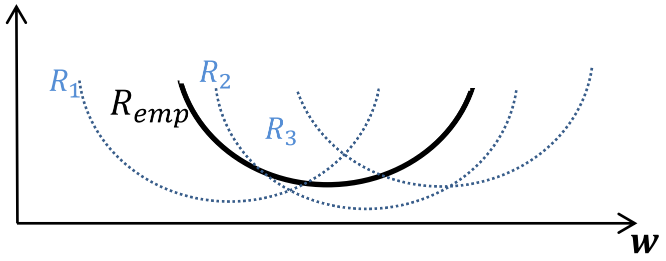
Pseudo-inverse and gradient descent:
Same error criterion → gradient descent tends to the same solution.

Pros & cons
- does not require matrix inversion
- still needs all input-output pairs from D_N

Training: one sample stochastic gradient descent (1-sample SGD)

If data are *stationery*, **then** minimizing R_{emp} is equivalent to successively minimizing each R_{emp}^k

$$R_{emp} = \frac{1}{2}\sum_{i=1}^{N}(h(x_i) - y_i)^2 = \frac{1}{2}\sum_{i=1}^{N} R_i$$



The formula for the update of linear regression models in the (stochastic) gradient descent becomes:

$$w(t+1) = w(t) + \alpha(y_k - w(t)^T\tilde{x}_k)\,\tilde{x}_k$$

Only one observation is used at each step k.

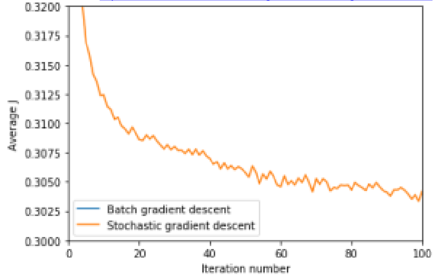
Difference between i, k :
- i and k are indices on the observations (1…N)
- i is the index in the database of observations, while k identifies the order of presentation, which my differ. The difference between i and k is not crucial here.

1-sample stochastic gradient descent

```
Initializes $w(0)$
Repeat
    Select $(x_k, y_k)$ from $D_N$

        - Forward pass
          Compute output
          $h(x_k) = w(t)^T\tilde{x}_k$

        - Backward pass
          Compute gradient
          $\nabla_w R_k = (h(x_k) - y_k)\,\tilde{x}_k$

        - Update parameters
          $w(t+1) = w(t) - \alpha\,\nabla_w R_k$
```

Training: Mini-batch gradient descent

Credit: https://adventuresinmachinelearning.com/stochastic-gradient-descent/



Note about 1-sample SGD: As we are only considering *one* example at a time, the cost will fluctuate based on the training examples and it will not necessarily decrease. But in the long run, you will see the cost decrease with the fluctuations.

Mini-batch gradient descent is a trade-off between 1-sample SGD and the BGD.

In mini-batch gradient descent, the cost function (and therefore gradient) is averaged over a small number of samples (e.g., 10 - 500).

This is opposed to the SGD batch size of *1* sample, and the BGD size of *all* the training samples.

The default **batch_size** in Keras is 32

Note: in many sources, the terms *mini-batch* and *stochastic gradient descent* are synonymous.

Question for you!

Models are rarely optimised using batch gradient descent (BGD) and instead we use mini-batches from the dataset to train on (often implemented under the name Stochastic Gradient Descent, or SGD). What are the advantages of doing this?

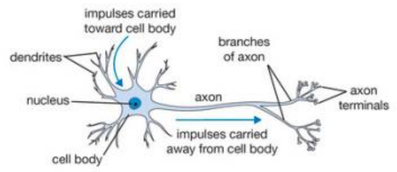
Select all that apply.

A. Using BGD is very slow for large datasets, as we need to iterate over the entire dataset before making a single parameter update.
B. SGD is able to jump out of local minima that would otherwise trap BGD.
C. SGD typically converges in fewer iterations.
D. SGD gives a better approximation to the true gradient.

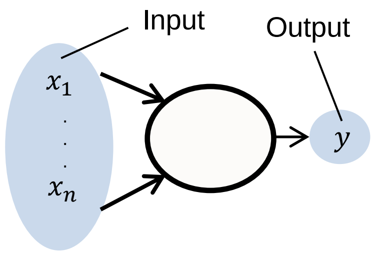
Perceptron

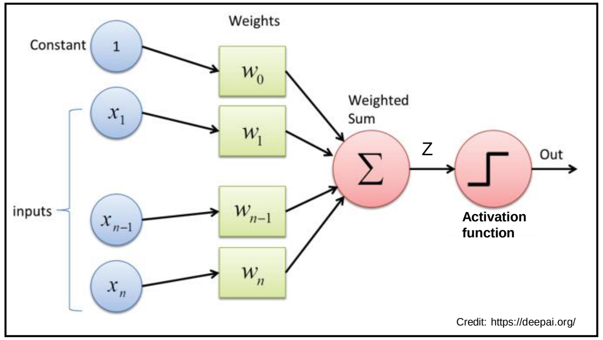
- Probably the most elementary way to perform binary classification
- The basic computing unit in neural networks : ‘neuron’
  - Example: Perceptron
- A **perceptron** ≈ a neural net with only **one neuron**.

Biological neuron


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

From Stanford cs231n lecture notes





Credit: https://deepai.org/

Vocabulary:

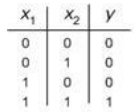
- $f()$ is the **activation function** of the neuron.
- $(w_0, w_2, \dots, w_n)^T$ are the **synaptic weights**.
- w_0 is also called the **threshold** (Bias value).

During the **learning** process, we search the optimal values of the **synaptic weights.**

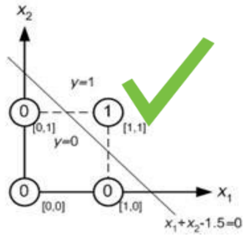
$$z = \sum_{j=1}^{n} w_j\, x_j + w_0 = w^T \tilde{\boldsymbol{x}}$$

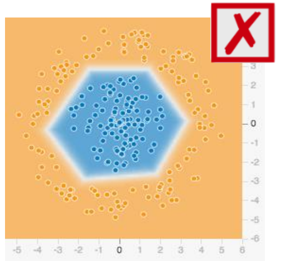
$$out = f(\mathrm{z}) = \begin{cases} 1\ if\ z \geq 0 \\ 0\ if\ z\ < 0 \end{cases}$$

It is a linear separator



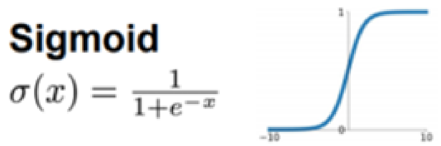
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1





Activation function

- The activation function takes as input the weighed sum of the input (i.e., $\mathrm{z_i} = w^T \tilde{x_i}$).
- The step function $f(\mathrm{z}) = \begin{cases} 1\ if\ z \geq 0 \\ 0\ if\ z\ < 0 \end{cases}$ was originally used as activation function in the perceptron.
- Note that
  - if $f(\mathrm{z}) = z$ is used as activation function then the *perceptron* becomes *linear regression*
  - if sigmoid $\sigma(z)$ is used than the *perceptron* becomes *logistic regression*

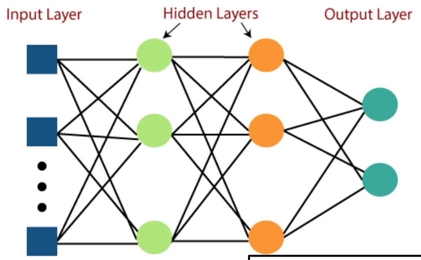


Multilayer perceptron

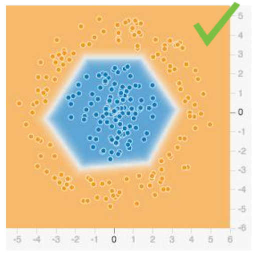
Multilayer perceptron (MLP)
(also called *Feed-forward neural networks* or *Plain vanilla*)

- Combine input information in a complex & flexible neural net model

- Network Structure
  - Multiple layers
    - Input layer
    - Hidden layer(s)
    - Output layer



Warning: this representation don't show the bias values!

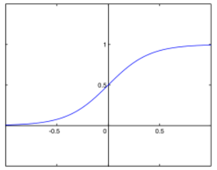


- Multilayer perceptron → fully connected layers

Question for you!

Why is it needed to add non linear activation function?
What if all activation functions are linear?

- Activation functions f in the hidden layers introduces nonlinearity.
  - Historically, the most common activation functions in the hidden layers was sigmoid.



$$f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

← Old school! For hidden layers in MLP use ReLu

$\sigma(z)$ transform any input $z \in \mathbb{R}$ into a number in $[0, 1]$.

Activation function

Some other popular examples of activation functions

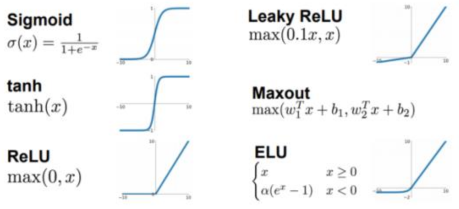


Image taken from Stanford's CS231n class.

But… How to choose the activation function that we will use in our deep learning model ??

There are no general rules but some *common* rules depending on the type of layers (input layer, hidden layer(s) and output layer)

→ For the input layer, it's simple: As input layers takes raw input from the domain, there are no activation functions in the input layer.

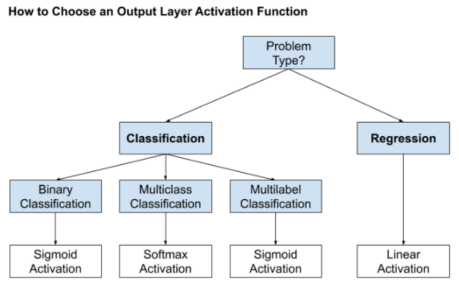
Activation for Hidden Layers

There might be three activation functions that you might want to consider using in hidden layers; they are:

- Logistic (Sigmoid)
    - Traditionally, the sigmoid activation function was the default activation function in the 1990s.
    - Still used in some advance cases (e.g., recurrent neural network)
- Hyperbolic Tangent (Tanh)
    - Still used in some advance cases (e.g., recurrent neural network)
- Rectified Linear Activation (ReLU)
    - *"In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU (…)"* I. Goodfellow et al. **Deep Learning,** 2016

A MLP neural network will almost always have the same activation function in all hidden layers.

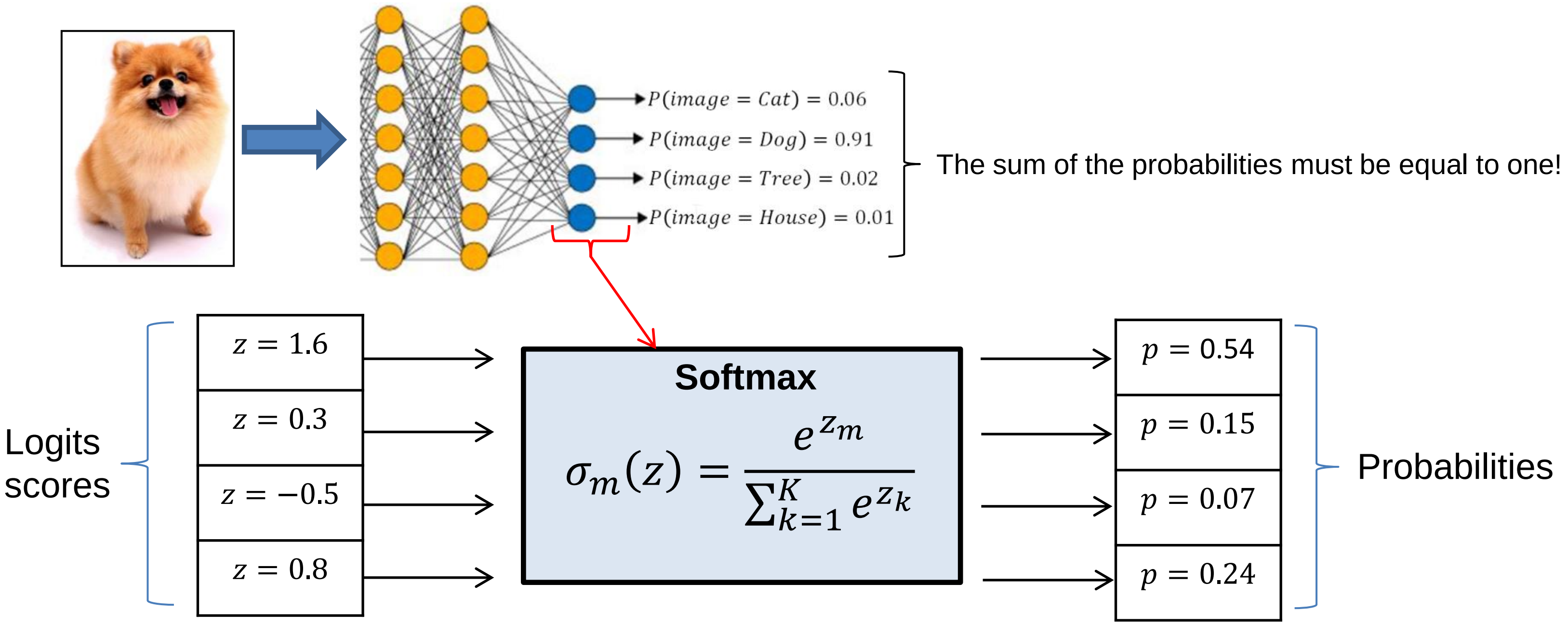
Activation for Output Layers



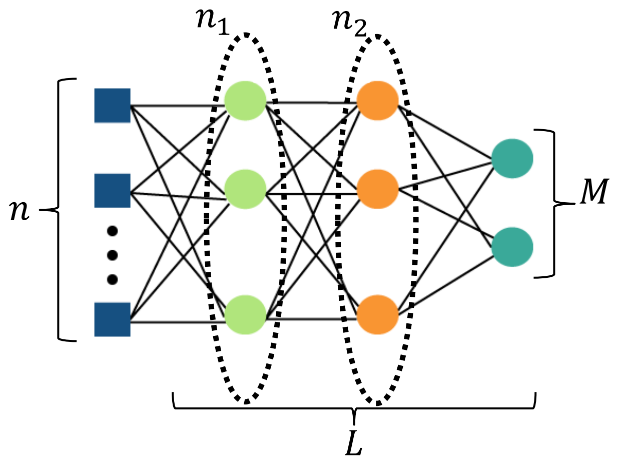
Credit: machinelearningmastery.com/

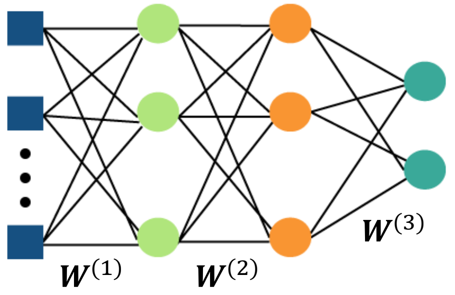
Activation function – Softmax

- ***Softmax*** is a kind of generalization of the logistic function for the *multiclass* classification problem.
- In a *multiclass* classification problem:
  - the target variable can take many distinct discrete values : $y^i \in \{C_1 \ldots C_K\}$ where K is the number of class.
  - The output of the model $h(\boldsymbol{x})$ is a vector of values in [0,1] which sum to 1 (i.e. probabilities)



Multilayer perceptron – notation





- n : number of input features.
- $n_1, n_2, \dots, n_l, \dots$: number of nodes in layers
- M : number of output values
- L : number of layers.

$\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times (n_{l-1}+1)}$: synaptic weights between layers $(l-1)$ and l.

Example, if $n = 1$ and $n_1 = 3$ then

$$\boldsymbol{W}^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{pmatrix} \in \mathbb{R}^{3\times 2}$$

Questions for you!

Draw the following neural network:

- $L = 3$
- $n = 1$
- $n_1 = 2$
- $n_2 = 3$
- $M = 1$

How many synaptic weights (tips: don't forget the bias values ;-))?

A. *12* synaptic weights
B. *17* synaptic weights
C. *49* synaptic weights
D. *15* synaptic weights

How many trainable parameters does a feedforward network have with input shape (64,), three hidden layers with 16 units each and a final linear layer with 8 units?

- 1720
- 968
- 7416
- 142

Why do we use non-linear activation functions (such as *tanh* or *relu*) in neural networks?

A. To allow the usage of higher learning rates, thus speeding up the convergence during the optimization.
B. To induce sparse connectivity in the network weights.
C. Without non-linear activation functions, the network would only be able to model linear functions of the data.
D. So that the model activations are equivariant with respect to the input features