Machine Learning

Daniel Glez-Peña &
Hugo López-Fernández
@SINGgroup
www.sing-group.org

Agenda

- Machine learning Introduction
  - Supervised learning
  - Unsupervised learning
  - Machine Learning in Bioinformatics

- Supervised learning
  - Workflow
  - Overfitting
  - Feature selection
  - Classifier evaluation and comparison
  - Algorithms
    - Naïve Bayes
    - Instance-based
    - Decision trees
    - Linear

Machine Learning Introduction

What is Machine Learning?

Trying to make machines able to solve problems without explicitly programming them

"Training" instead of programming

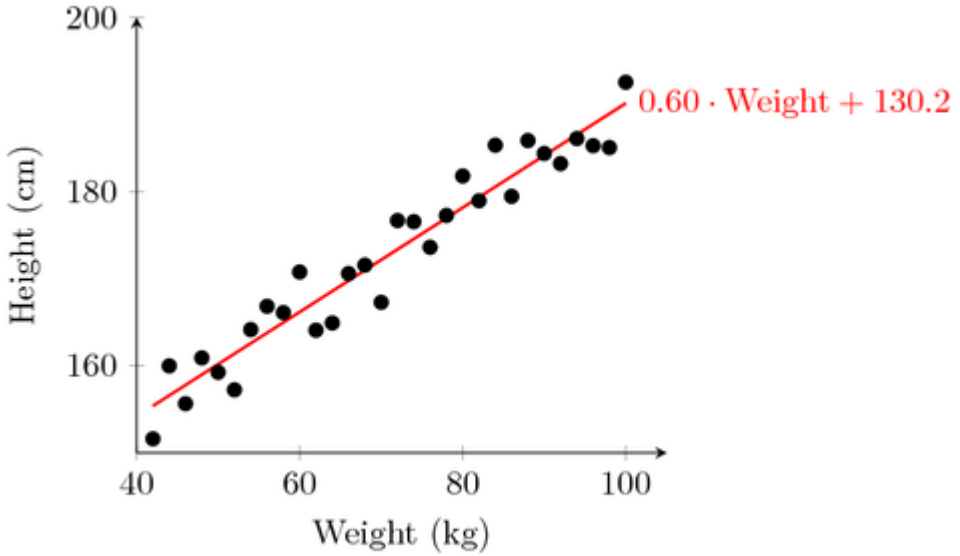
Types

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

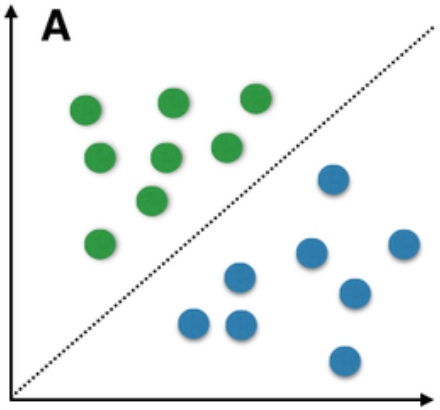
Supervised learning

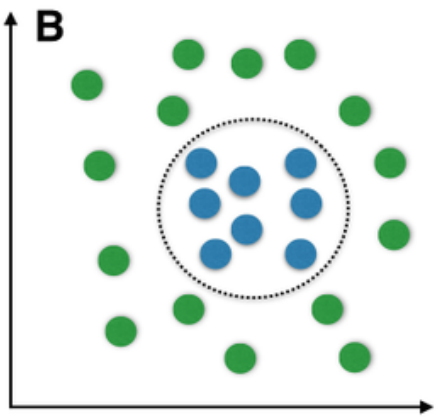
Given a set of "instances", each one with a set of measured "attributes" and an "outcome" value, we want to train a "model", that predicts the outcome in further problem instances.

Regression (outcome is a number)



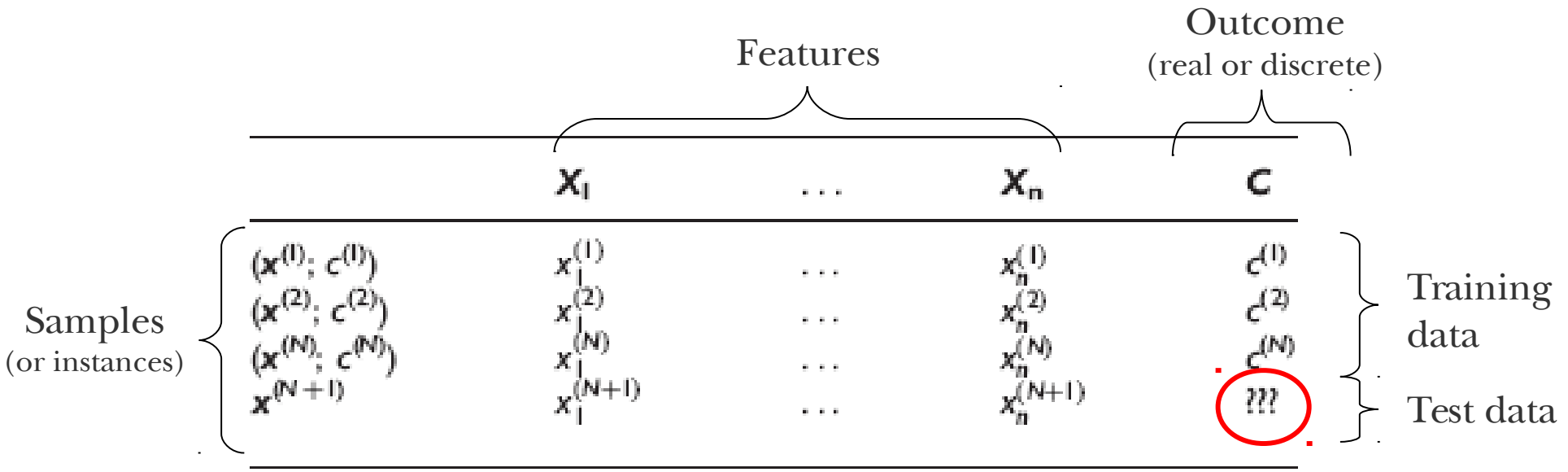
Classification (outcome is a category)





Supervised learning

Input dataset structure for learning



Unsupervised learning

Learning underlying structures in data

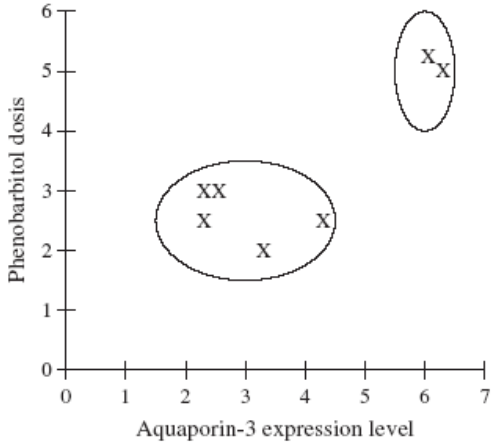
- Clustering (learn group of samples)
  - Partition a set of "instances" in several groups (clusters) given the differences between them
    - Their are based on "distances" between instances that is a problem-dependant issue
      - Typical: Euclidean, Pearson, Sperman
- Dimensionality reduction (learn alternative representations)
  - Reduce the number of input variables while trying to keep the maximum information of the dataset
    - Very useful for visualization (n dimensions → 2D or 3D)

Unsupervised learning

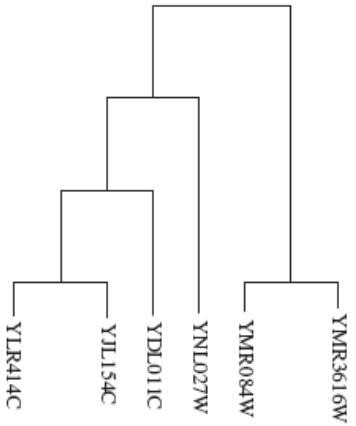
Popular techniques

- Partition clustering
  - k-means, SOM, GCS, PAM
- Hierarchical clustering with single-linkage, complete linkage, centroid linkage and wards-criterion
  - They produce the popular “dendograms”
- Model-based clustering

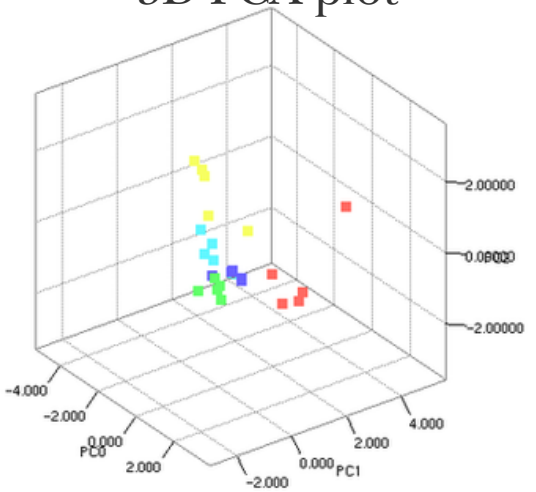
k-means



Hierarchical clustering



3D PCA plot



Bioinformatics

Application of the Information Technologies to the field of Molecular Biology

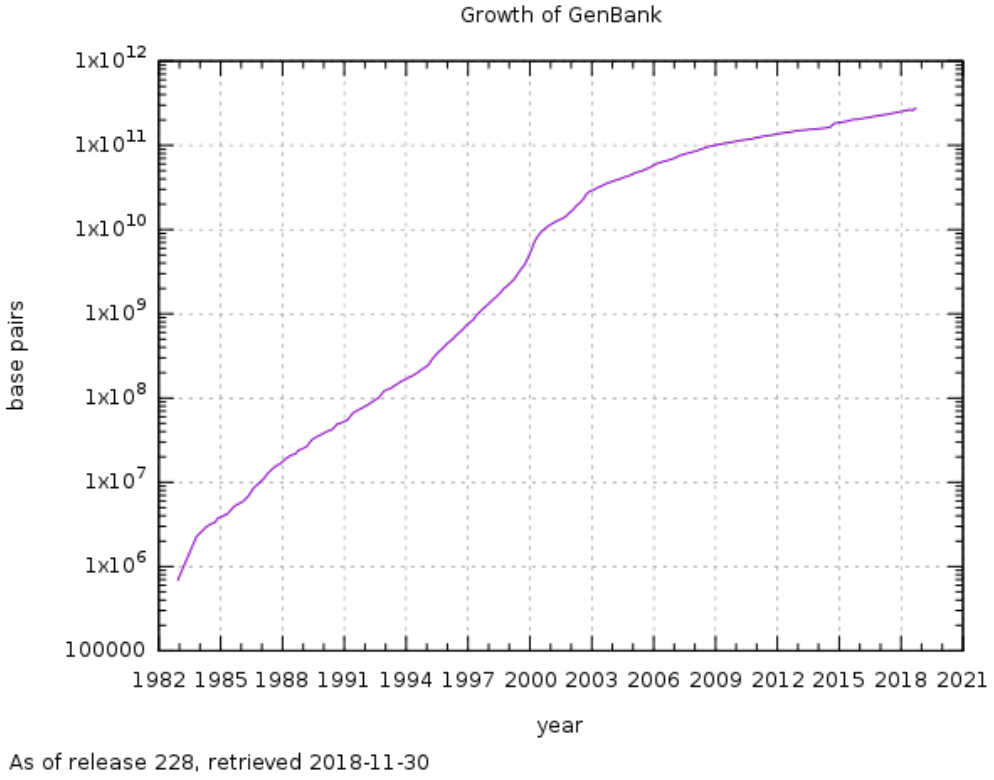
Databases

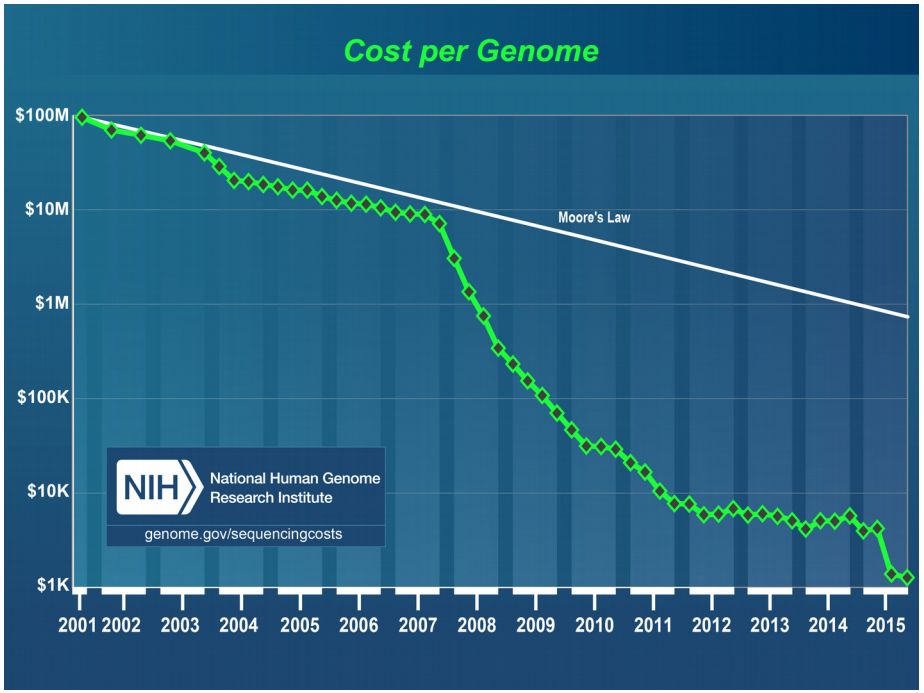
Algorithms

Statistical techniques

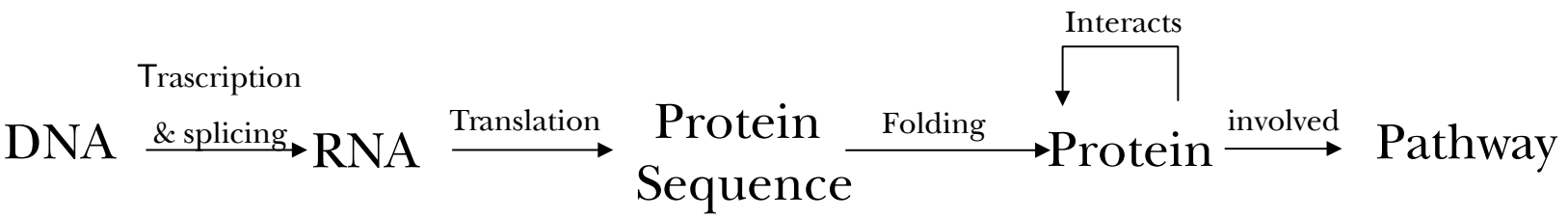
… to solve formal and practical problems arising from the management and analysis of biological data

Bioinformatics





The Dogma & Bioinformatics



Nucleotide sequence	Transcripts	Aminoacid sequence	3D structure	Set of Reactions
ACTTGTC ATGGCGA CTGTCCT TTGTGC …		MEEPQSDPSV EPPLSQETFSD LWKLLPENNV LS…		

Interactomics

GENOMICS	PROTEOMICS	METABOLOMICS
Sequence analysis Genome annotation Analysis of mutations [DNA-seq] Gene expression analysis [microarrays/RNA-seq] **Evolution** Phylogenetic reconstruction Comparative genomics	Protein expression analysis [mass spectometry] Protein interaction prediction [3D docking] Protein structure prediction [folding]	Modelling biological systems Functional analysis

Bioinformatics Data

Genomics

Sequences

Entrez Nucleotide

EMBL NUCLEOTIDE SEQUENCE DATABASE

DDBJ DNA Data Bank of Japan

Genomes

Entrez Genome

e! Ensembl

UCSC

TIGR

Gene-centric

Entrez Gene

GeneCards®

Proteomics

Proteins

swissprot

TrEMBL TRANSLATED EMBL

Entrez Protein

UniProt the universal protein resource

PIR

Structure

PDB PROTEIN DATA BANK

SCOP STRUCTURAL CLASSIFICATION OF PROTEINS

CATH Protein Structure Classification

Domains

prosite

InterPro

Conserved Domains

Pfam

SMART

Interactomics & Metabolomics

Prot-Prot interactions

STRING

IntAct

Pathways

KEGG

BIOCARTA

PathwayInteractionDatabase

GenMAPP Gene Map Annotator and Pathway Profiler

Ontologies

the Gene Ontology

MeSH

Bibliome

PubMed

Experimental data

GEO Gene Expression Omnibus

ArrayExpress

Supervised learn. in Bioinformatics

- Genomics
  - Gene finding (if a sequence is a coding region)
  - Splice site prediction (if a sequence is a splice site)
  - Predict disease genes (e.g: from its sequence length)
  - Prediction of mutation (SNP) effect
  - Cancer prediction from gene expression (microarrays/RNA-seq)
- Proteomics
  - Prediction of secondary structure (alpha-helix, beta-sheet, etc.)
  - Prediction of sub-cellular location of the protein
  - Cancer prediction from protein expression (mass spectra)

Supervised learn. in Bioinformatics

- Systems biology
  - Predict the cell migration speed (high, low) from the phosphorilation levels of signaling proteins
  - Predict a gene regulatory level (up-regulated or down-regulated given the 'related' genes expression)
- Text mining
  - Protein/gene recognition in biomedical literature
    - is this word a gene/protein given some word features: ortographic, part-of-speech, suffix, trigger words, etc…??

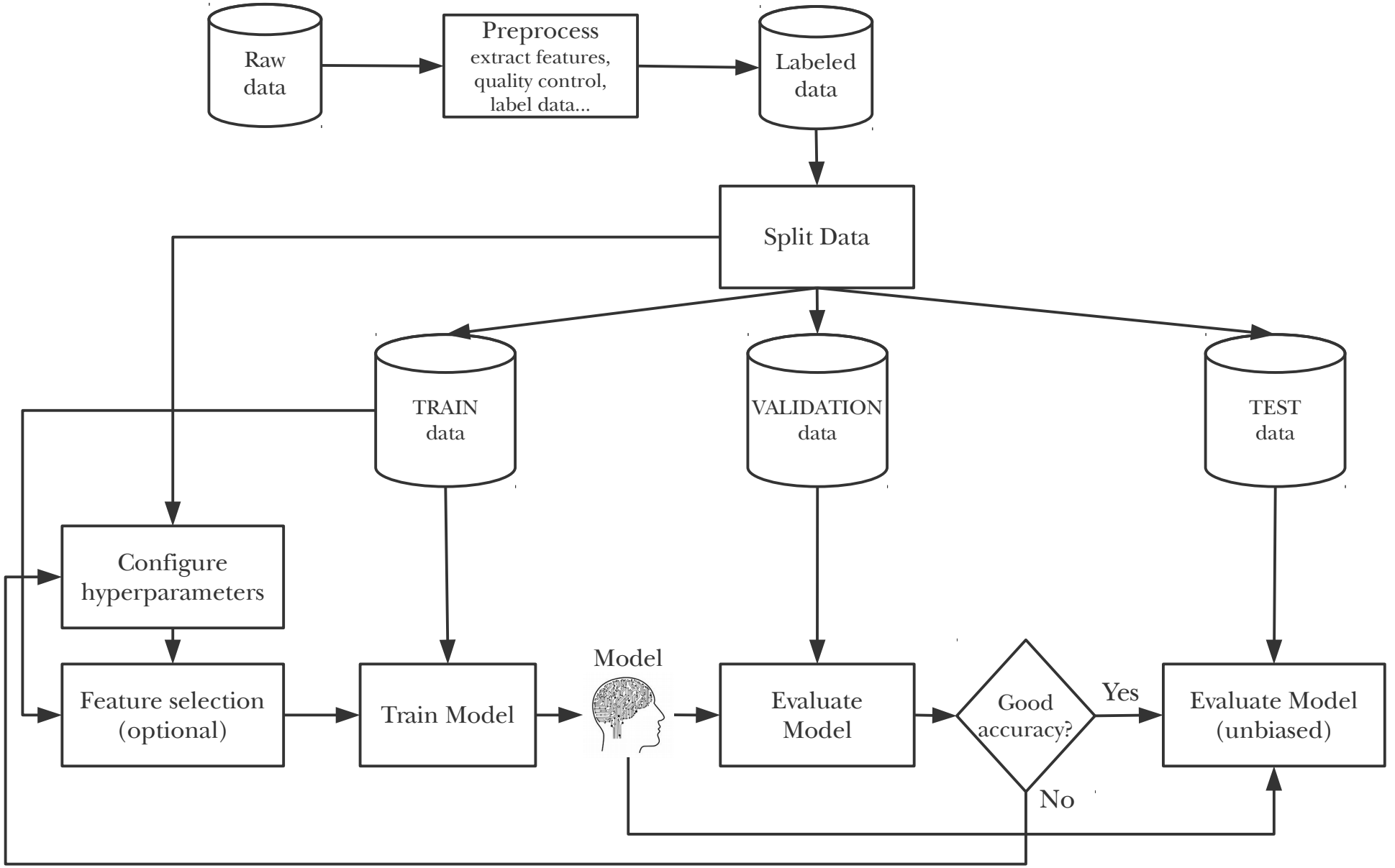
Unsupervised learn. in Bioinformatics

Mainly in Genomics and Proteomics

- Co-expression detection (group of genes/proteins with similar expression)
- Subclass discovery (group of samples given the expression of its genes)
- Expression data visualization with dendograms

Supervised Learning

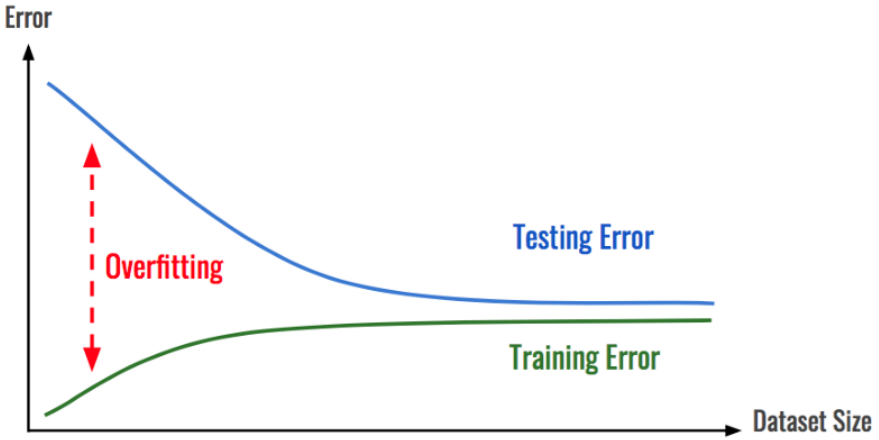
Supervised learning workflow

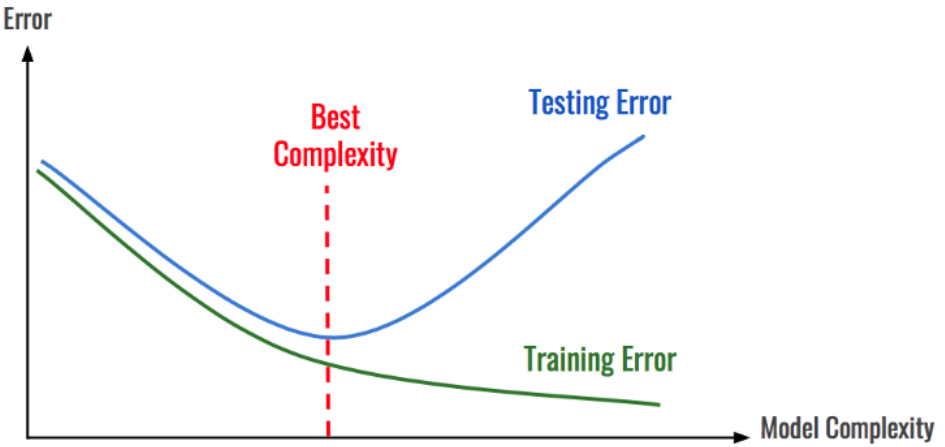


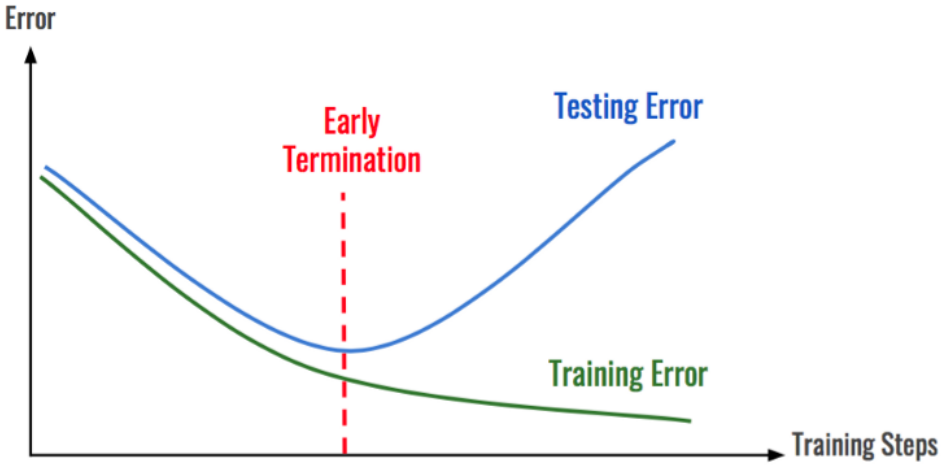
Overfitting

https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42

Overfitting







https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42

Feature selection

Are all input attributes useful?

Advantages: reduced cost in data acquisition, more interpretable models, faster training, and better accuracy

It is a search space problem (2^n-1), in general:

1. Generate a subset
   - Brute force (all possible subsets)
   - Deterministic/not deterministic heuristic search
2. Evaluate subset
   - Statistical estimation: Information Gain, X2, t-test, DFP, CFS
   - Wrapper (use classifier accuracy in training set)
3. if (!halt_condition) GOTO 1

Classifier evaluation and comparison

Classifier evaluation

Performance metrics. In binary classification:

Predicted \ Actual	Positive	Negative	
Positive	True Positives (TP)	False Positives (FP) Type-I error	Positive Predictive Value (PPV) or precision = TP / (TP + FP) False Discovery Rate (FDR) = 1 - PPV
Negative	False Negatives (FN) Type-II error	True Negatives (TN)	Negative Predictive Value (NPV) = TN / (FN + TN) False Omission Rate (FOR) = 1 - NPV
	Sensitivity or recall or True positive rate (TPR) = TP / (TP+FN) False Negative Rate (FNR) = 1 - TPR	Specificity or True Negative Rate (TNR) = TN / (FP+TN) False Positive Rate (FPR) = 1 - TNR	Accuracy = (TP + TN) / N

Classifier evaluation

Performance metrics. In multi-class classification:

Accuracy and kappa are valid for multi-class classifier evaluation

We can also use the previous metrics for each class by considering

positive = class

negative = other classes

However, there will be multiple sensitivities, specificities, PPV, etc. (one per class)

Classifier evaluation

Ideally, we should do several tests for a model in order to better estimate its real performance. If the test data is big enough, and not biased, the estimation is good

However, when we have a reduced number of samples, it is not easy. There are several validation frameworks:

- Holdout
- Bootstrapping
- Cross-validation

Classifier evaluation

Holdout

- Divide dataset randomly in several subsets
- In general, we take 2/3 for train and the rest for test
- Variant: “resampling”
  - It consists in conducing several holdout test over different train and test datasets

Classifier evaluation

Bootstrapping

Given a dataset with N samples

Create a training set of size N, taking samples N times randomly, with replacement

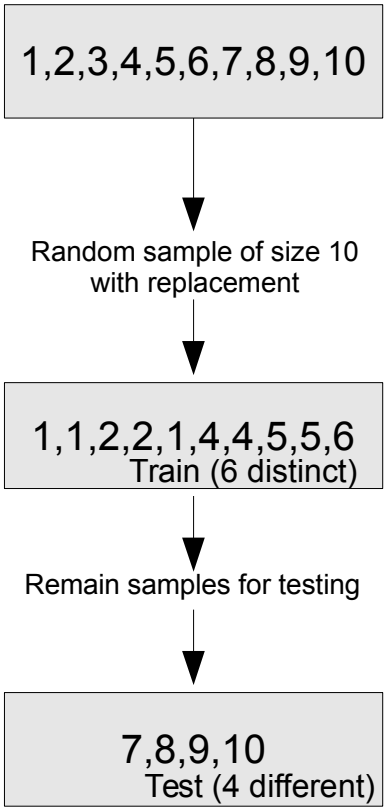
In general, with a size big enough, the number of different instances in the sample tends towards 63.2% of N

$$P(\text{of being selected with 1 trial}) = \frac{1}{N}$$

$$P(\text{not being selected}) = 1 - \frac{1}{N}$$

$$P(\text{not being selected in N trials}) = \left(1 - \frac{1}{N}\right)^n \approx e^{-1} = 0.368$$

$$\text{Number of different samples on train} = 1 - 0.368 = 0.632$$



The final error of this method can be weighted using the training error:

$$e = 0.632 \cdot e_{\text{test}} + 0.368 \cdot e_{\text{train}}$$

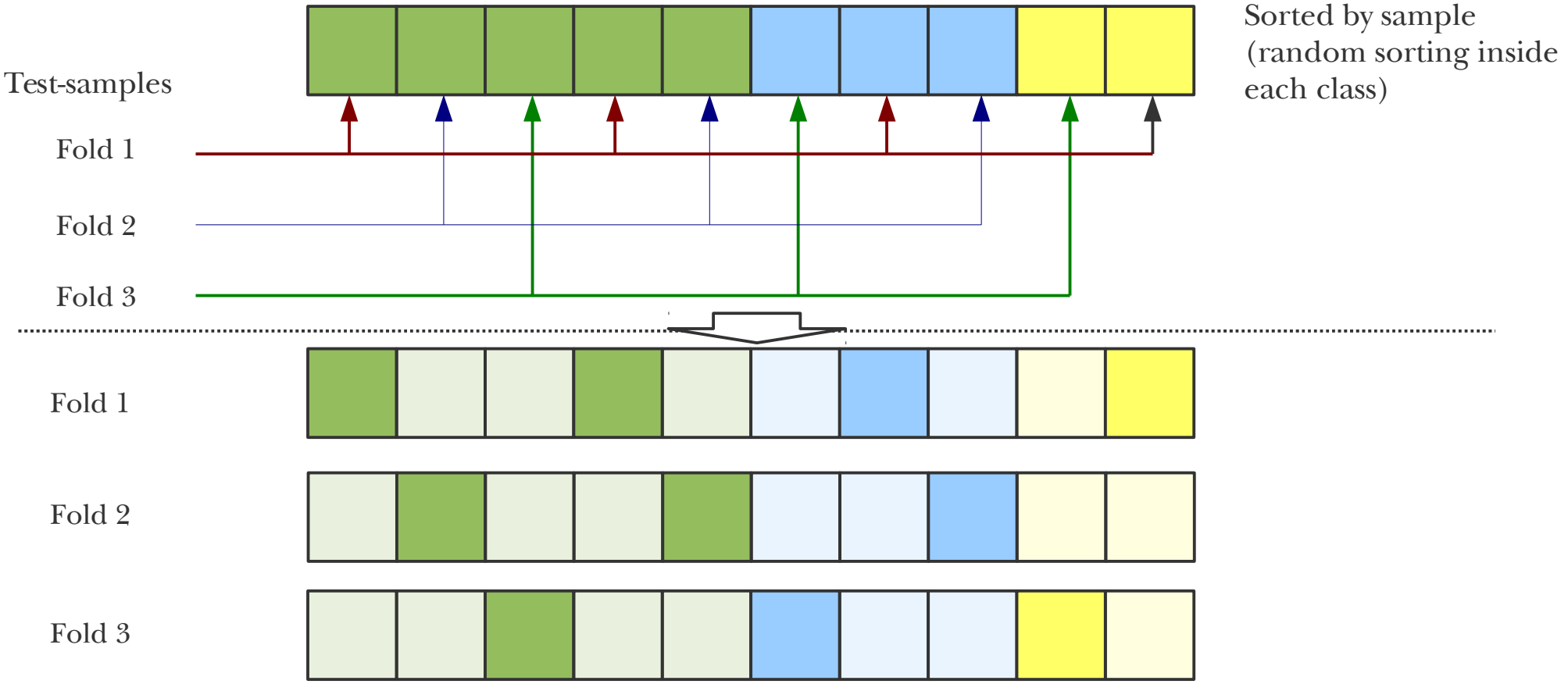
Classifier evaluation

- Cross-validation (CV)
  - Create K pairs of training and test
    - Example. With 10 samples, conduct a 3-fold CV, we get 3 pairs, with a 33% of samples for testing and the rest for training
  - Each sample appears as test sample in exactly one pair
  - K=10 is one of the most popular value
  - To better estimate the performance, we can repeat CV 10 times, giving a total of 100 train-test experiments
  - Stratified: keep the same class distribution of the complete dataset in each train/test subsamples
  - Leave-one-out. Variant where K=N, that is, the test subset in each pair has only one sample
    - Useful when there are very few training samples

Classifier evaluation

Cross validation.

Example. Stratified 3-fold CV



Classifier comparison

- We can use the CV error directly
  - However, differences can be due to chance
  - We can employ statistical tests over CV estimations to tell if differences are statistically significant. Example methods
    - McNemar test
    - CV + Paired t-test
    - 5x 2-fold-CV + Paired t-test
    - Repeated CVs + corrected resampled t-test (weka)

Classifier comparison

McNemar test

Given the output of two classifiers (A,B) for each sample, we have 4 possibilities:

n_{00}: Both classifiers fail	**n_{01}: A fails, B hits**
n_{10}: B fails, A hits	n_{11}: Both classifiers hit

The next T score is based on observing differences between off-diagonal values (n_{10} y n_{01}), which are cells counting when classifiers differ.

Null hypothesis. Differences between n_{10} and n_{01} is 0.

$$T = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}$$

This score follows a chi-squared distribution with 1 degree of freedom. For example a $T > 3.8414$ gives p-values < 0.05.

Classifier comparison

CV + Paired t-test

Conduct a K-fold CV. We get K accuracies of each classifier

	Fold 1	Fold 2	Fold 3	Fold 4	...	Fold K
Classifier A	0.7	0.75	0.78	0.79		0.77
Classifier B	0.6	0.65	0.64	0.62		0.69

Null hypothesis: both classifiers has the same mean accuracy

Conduct a Student's paired t-test

Problem: The t-test assumes independency of each pair
However, CV generates overlapping training samples

Classifier comparison

5x 2CV + Paired t-test

Conduct 2-fold CV 5 times

The first replicate to estimate error; all replicates to estimate variance

On each of the 5 experiments:

	Fold 1	Fold 2
Classifier A	0.7	0.75
Classifier B	0.6	0.65

$p^{(1)}$: differences of A and B errors in fold 1

$p^{(2)}$: differences of A and B errors in fold 2

$$s^2 = (p^{(1)} - \bar{p})^2 + (p^{(2)} - \bar{p})^2$$

Compute an statistic that follows a distribution similar to Student's t with 5 degrees of freedom:

$$\tilde{t} = \frac{p_1^{(1)}}{\sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}}$$

i = iteration

With t < 2.02 we get p-values < 0.05

Classifier comparison

Repeated CVs + corrected resampled t-test

Test included in Weka Experimenter

Conduct CV several times:

For example: 10x10-fold CV

We get 100 accuracy values for each classifier

Use a variant of Student's t-test which takes into account that samples are not independent (samples are repeated in different train-test samples) and also avoid the bias introduced with the total number of trials

$$t = \frac{\bar{d}}{\sqrt{\left(\frac{1}{k} + \frac{n_2}{n_1}\right)\sigma_d^2}}$$

$\bar{d}$: classifier A and B accuracy mean differences in each trieal (100 trials)
k : number of trials, in a 10x10CV, k is 100
n_1 : ratio for train , n_2 : ratio for test. In a 10CV n_1 is 0.9 and n_2 is 0.1

This score follows a Student's t distribution with folds-1 degrees of freedom (9 for 10-fold CV)

Supervised Learning Algorithms

Bayesian

Naïve Bayes

Classification consist in find P(class | features)

With conditional dependencies (using Bayes). We need to know 2^n different probabilities (n=number of features, unfeasible).

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)} \qquad P(cavities|pain\wedge infection)=\frac{P(pain\wedge infection|cavities)P(cavities)}{P(pain\wedge infection)}$$

With **conditional independence**

Idea: two random variables can be independent once a third variable is known.

Mathematically:

$$P(pain\wedge infection|cavities)=P(pain|cavities)P(infection|cavities)$$

pain and *infection* are independent once the value of *cavities* is known

Replacing in the previous Bayes formula:

$$P(cavities|pain\wedge infection)=\frac{P(pain|cavities)P(infection|cavities)P(cavities)}{P(pain\wedge infection)}$$

$$P(\neg cavities|pain\wedge infection)=\frac{P(pain|\neg cavities)P(infection|\neg cavities)P(\neg cavities)}{P(pain\wedge infection)}$$

Conditional indedepencence allows us to scale probability-based systems; indeed, they are more available than those assuming total independence

Bayesian

Naïve Bayes

Conditional independence (cont)

So we have:

effects(s)

$$P(C|F_1,\ldots,F_n) = \alpha P(C)\prod_{i=1}^{n} P(F_i|C)$$

cause

α=(1/denominator) in the Bayes function. It can be solved by considering that P(c) + P(¬c) = 1

It is a typical pattern: One cause has multiple effects, all of them conditionally independent, once such cause is known.

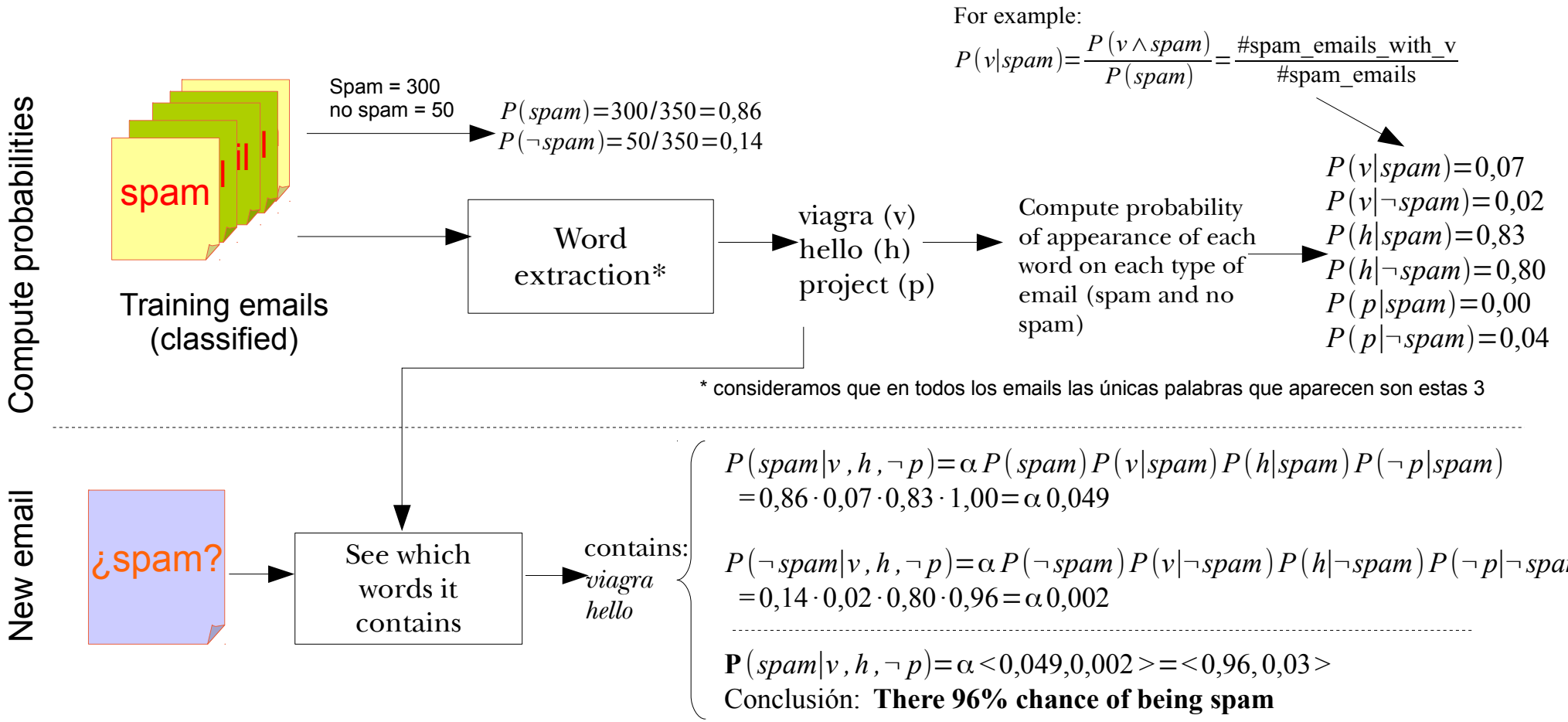
Naïve-Bayes (or simple-Bayes)

Sometimes, even when such conditional independence is not real, we make this assumption in order to make the system feasible

Bayesianos

Naïve Bayes

Naive-Bayes. Example: Detecting spam email



Instance based

Introduction

During training, we save all samples without any modification.

With a *distance function* we find the nearest saved sample to the new sample and we give the class of the found sample as prediction.

Distance functions:

Euclidean $$dist(p,q)=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}$$

Manhattan $$dist(p,q)=\sum_{i=1}^{n}|p_i-q_i|$$

Euclidean, with exponent >2 (increases the influence in greater distances).

Instance based

Normalization

If attributes have a different scale (for example, age [0, 100] and price [10k, 100k], attributes with larger values have bigger influence, so it is typical to normalize them, for example:

feature scaling:

$$scale(v_i) = \frac{v_i - min\, v_i}{max\, v_i - min\, v_i}$$

stardardization (mean 0, sd 1):

$$z(v_i) = \frac{x - \mu}{\sigma}$$

Non numerical attributes

The distance between two nominal values is 1 if they are different and 0 if they are the same

Decision Trees

Try to learn a decision tree from data

Basic iterative process:

- Select an attribute
- Create a branch for each value of the attribute
- Each branch includes a set of samples
- Repeat the process on each branch, taking only into account those samples under such branch
- If the samples are of the same class, stop

Decision Trees

Which attribute is the best?

We look for the attribute which gives more information about the class

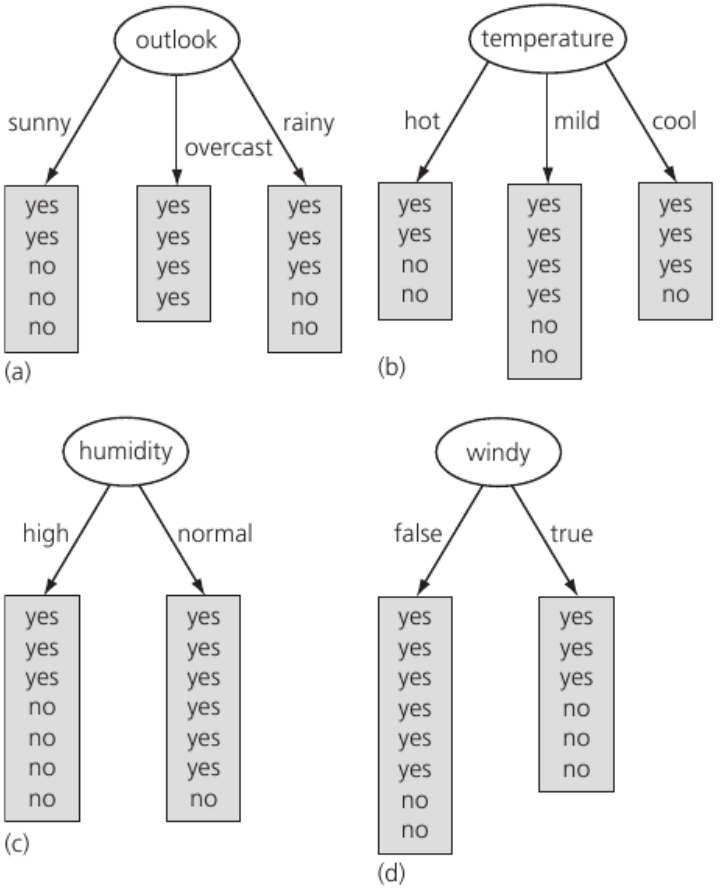
Information gain

We try that on each branch there is a dominant class

It is important to take into account all branches

Some branch can include samples of the same class but other branches don't

Branch-size weighted mean



outlook info 3 branches
info([2,3]) = 0.971 *sunny* branch
info([4,0]) = 0.0 *overcast* branch
info([3,2]) = 0.971 *rainy* branch
info(*outlook*) = info ([2,3], [4,0], [3,2]) = 5/14 * 0.971 + 4/14 * 0 5/14 *0.971 = 0.693

Root attribute: info(*root*) = info([9,5]) = 0.940
Gain(*outlook*) = info(*root*) – info(*outlook*) = 0.940 – 0.693 = 0.247 //best option
Gain(*temperature*) = 0.029 *Gain*(*humidity*) = 0.152 Gain(*windy*) = 0.048

Decision Trees

Function info([#classA, #classB, ..., #classN])

It should be:

- Multiclass
- If all are of the same class, it should return 0
- If all are distributed uniformly, it should be maximum
- It should meet the *step-by-step* evaluation property
  - Example (3 classes: A,B,C):
    - info([2,3,4]) = info([2,7]) + (7/9) * info([3,4])
    - First we decide A *vs.* "others" and then (in 7 of 9 cases) we decide if it is B or C
- An example of "info" function that meets these properties is *entropy*
  - entropy($[p_1,...,p_n]$) = $-p_1\log p_1$ - ... - $p_n\log p_n$
  - info($[p_1,...,p_n]$) = entropy($[p_1/N, ... , p_n/N]$), where $N = p_1+...+p_n$

Decision Trees

If we have attributes with many possible values, we should make a correction

- Many branches promotes gain, which is an undesired effect
- We compute *gain ratio*.
  - gain_ratio(attribute) = gain(attribute) / split_info(attribute)
  - where:
    - split_info(attribute) = info([#branch1, ... , #branchN])
      - Necessary info to know which branch will be taken. It only takes into account how much members belong to each branch. The more branches, the more split_info, so less gain

Outlook		Temperature		Humidity		Windy	
info:	0.693	info:	0.911	info:	0.788	info:	0.892
gain: 0.940–0.693	0.247	gain: 0.940–0.911	0.029	gain: 0.940–0.788	0.152	gain: 0.940–0.892	0.048
split info: info([5,4,5])	1.577	split info: info([4,6,4])	1.557	split info: info ([7,7])	1.000	split info: info([8,6])	0.985
gain ratio: 0.247/1.577	0.157	gain ratio: 0.029/1.557	0.019	gain ratio: 0.152/1	0.152	gain ratio: 0.048/0.985	0.049

Linear

Work with numerical attributes

They are intended to classify samples that are *linearly separable*

Linear regression

The predicted value for a sample $i = (a_1, ..., a_k)$ will be:

$$predict\left(a^{(i)}\right) = w_0 + \sum_{j=1}^{k} w_j \cdot a_j^{(i)}$$

w is a weight vector, which is computed by trying to minimize the training error ($x^{(i)}$ = class of sample i)

$$error = \sum_{i=1}^{n} \left(x^{(i)} - \left(w_0 + \sum_{j=1}^{k} w_j \cdot a_j^{(i)}\right)\right)$$

Linear

Linear Regression

How to compute *w*?

Gradient descent algorithm

Iterative algorithm to find the minimum of a function (the error function)
Briefly:

At each step, the error is computed and weights are updated in order to reduce this error.

The algorithm stops when the error is small enough or a maximum number of iterations is reached.

How it works? It is based on the derivative of the *cost function*

The cost function is similar to the error given the weights, but we want it to be positive (independent of the direction of the error) and easily derivable

$$J(w, w_0) = \frac{1}{2} \sum_{i=1}^{n} \left(x^{(i)} - \left(w_0 + \sum_{j=1}^{k} w_j \cdot a_j^{(i)} \right) \right)^2$$

Linear

Linear Regression

Gradient descent algorithm (cont)

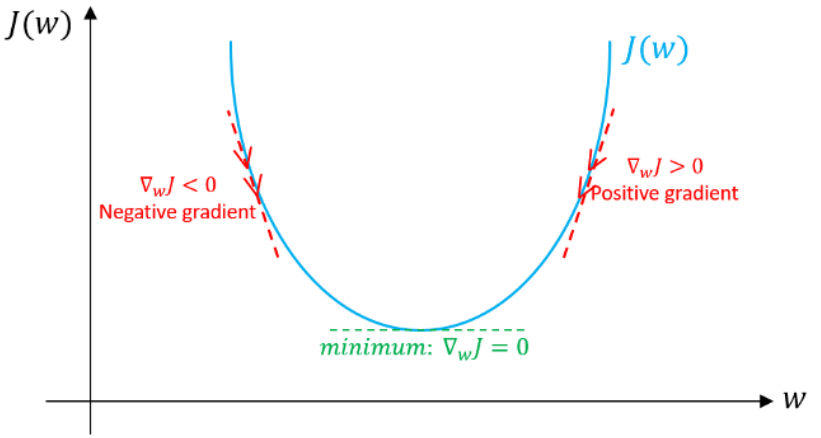
This algorithm needs the *gradient,* based on the *derivative* of the *cost function*

$$\frac{\partial J(w, w_0)}{\partial w_f} = -\sum_{i=1}^{n} \left(x^{(i)} - \left(w_0 + \sum_{j=1}^{k} w_j \cdot a_j^{(i)} \right) \right) a_f^{(i)}$$

Intuition: the derivative means how the cost varies when changing the weight w_f. It depends both on the amount of current error and how big are the values of the attribute in the training set.

At each iteration, each weight w_f is updated in the *opposite direction of its partial derivative and at a learning rate* α

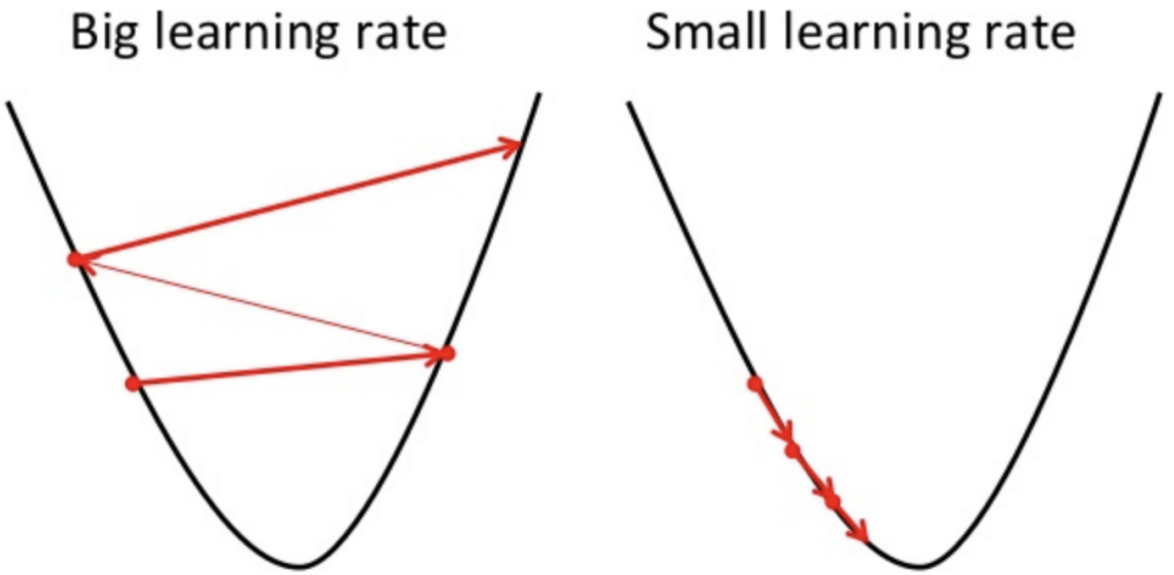
$$w_f := w_f - \alpha \frac{\partial J(w, w_0)}{\partial w_f}$$



Linear

Gradient descent algorithm (cont)

- The learning rate α must be carefully selected
  - Very small learning rates makes the training process slower
  - Very large learning rates makes the descent to jump over the minimum



Linear

Linear Regression

Adapting linear regression for classification seems to be easy: each data point is associated with 0 or 1, for each of the two possible classes

Problems:

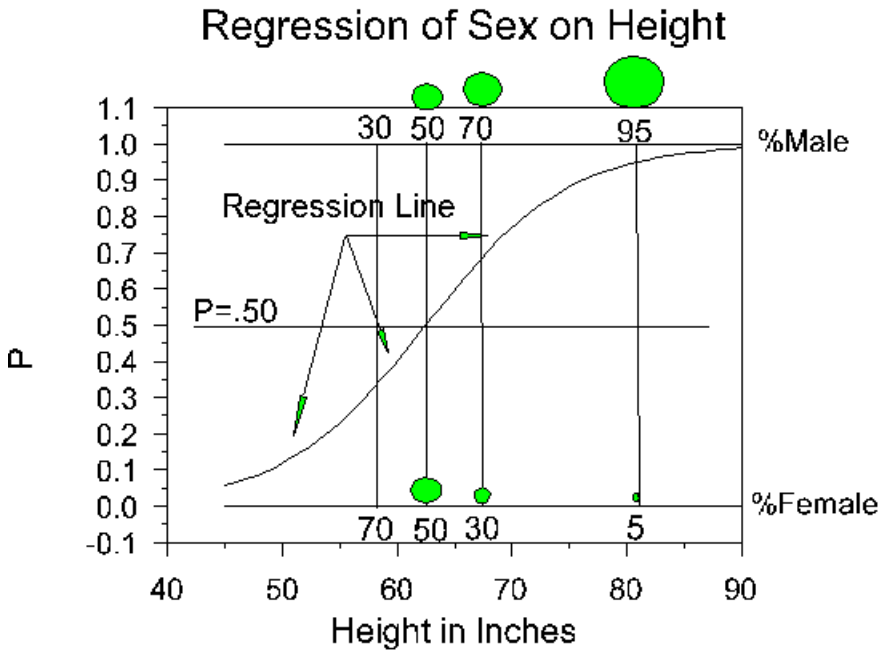
Linear regression does not gives values between 0 and 1, which are intuitively associated to a probability of belonging to a class.

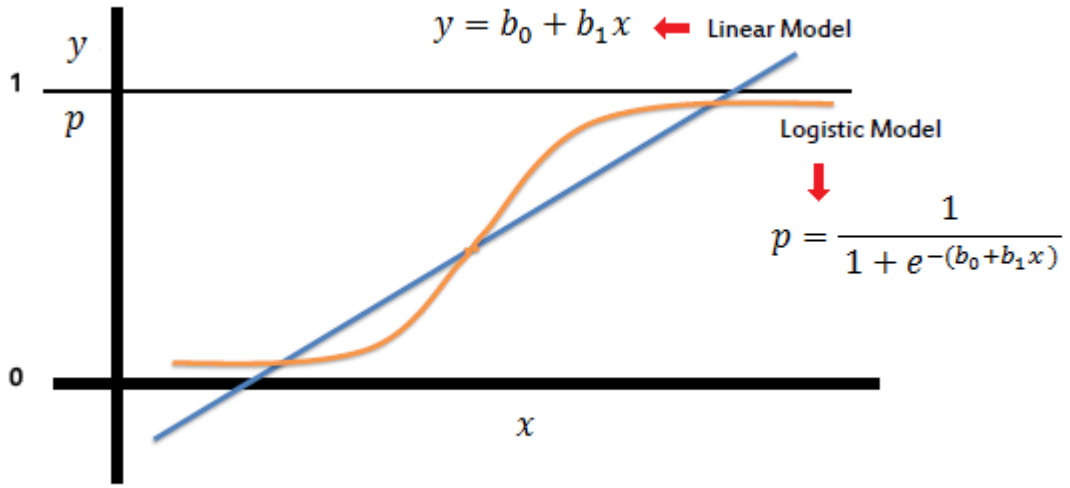
Not statistically sound: it assumes that error follows a normal distribution and the variance is constant, which is not true for binary outputs.

Linear

Logistic regression

Graphical interpretation of the outcome





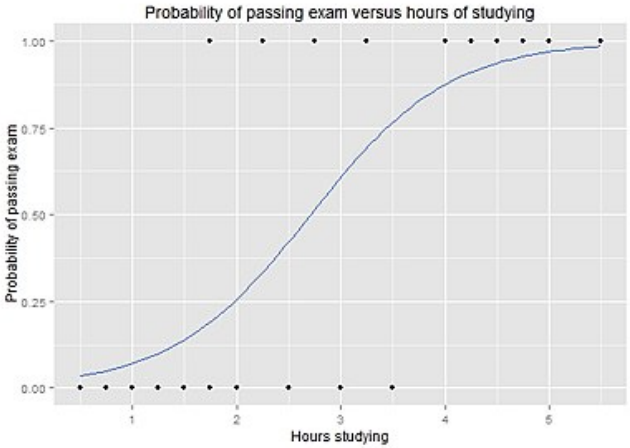
Linear regression vs. Logistic regression

Linear

Logistic regression

Gives output between 0 and 1. In bi-class, it can be assumed as the probability of belonging to class 1.

$$Pr(1|a^{(i)}) = \frac{1}{1+e^{-(w_0+w_1 \cdot a_1+\ldots+w_k \cdot a_k)}}$$



The cost function is different and based on logarithms, which gives high cost for confident wrong predictions:

$$J(w, w_0) = -\sum_{i=0}^{n} (1-x^{(i)}) \boxed{\log(1-Pr[1|a_1^{(i)}, a_2^{(i)}, \ldots, a_k^{(i)}])} + x^{(i)} \boxed{\log(Pr[1|a_1^{(i)}, a_2^{(i)}, \ldots, a_k^{(i)}])}$$

Linear

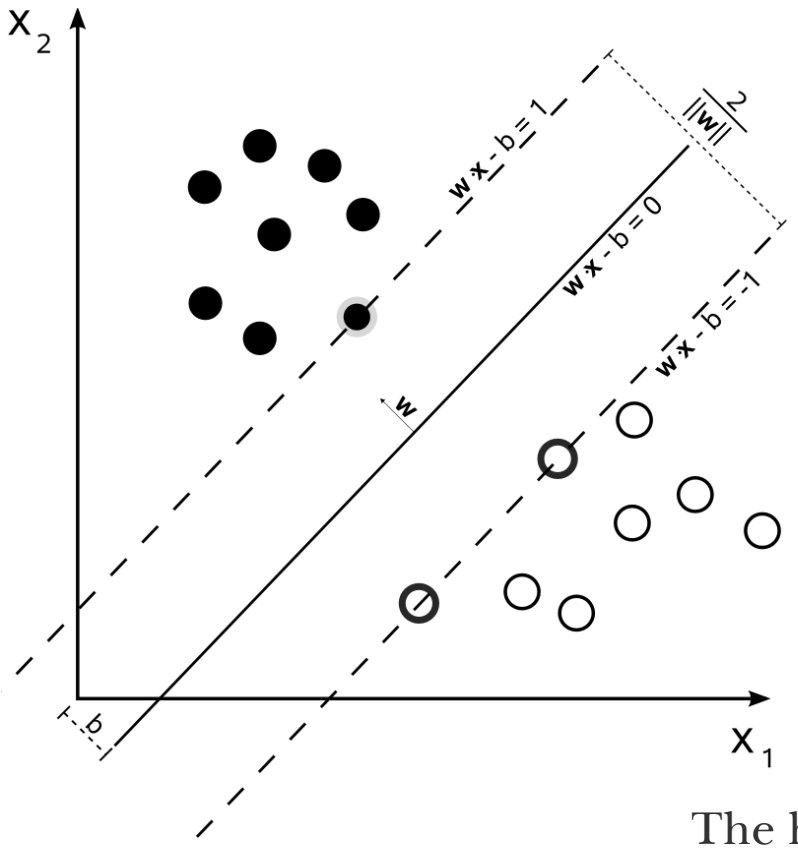
Logistic regression

- For multiclass (K classes), there are several methods
  - Take a “pivot” class (say class K)
  - Do a regression for each other class (K-1 classes) against the pivot class (class K)
  - Adjust probabilities to sum up 1
  - Choose the class with major probability

Linear

Support Vector Machines (SVM)

Binary classifier (easily extensible to multiclass)



Samples are represented in n-dimensional space (n = no. features)

Learn:

$$y = f(x),\ y \in \{\pm 1\},\ x \in \mathbb{R}^n$$

The objective is to find an hyperplane (n-1 dimension) which separates positive from negative instances.

Mathematically

Large margin

Find (w, b), minimising: $\|w\|$

subject to: $y_i(w \cdot x_i - b) \geq 1$

The hyperplane separates each class instances

Linear

Support Vector Machines (SVM)

Once w y b are found, classification is given by:

$$f(x) = w \cdot x + b$$

Giving the value +1 if $f(x) > 0$, or -1 otherwise

Linear

Support Vector Machines (SVM)

If not all samples can be separated.

Soft-margin SVM. Allows an error that should be minimized. C parameter indicates the cost of an error during training.

Now the objective is:

$$\text{Find } (w, b), \text{ minimizing: } \|w\| + C \cdot \sum_{i=1}^{m} \xi_i$$

$$\text{subject to: } y_i (w \cdot x_i - b) \geq 1 - \xi_i, \xi_i \geq 0$$

Linear

Support Vector Machines (SVM)

Non linear case

Alternative: Map input data to a higher dimensional space and try to find an hyperplane in that space

preprocess data:

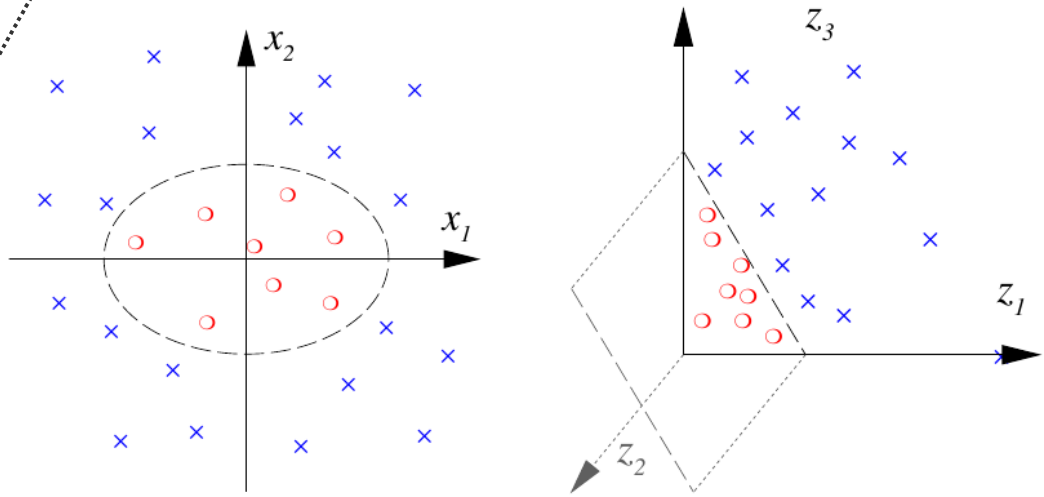
$$x \mapsto \Phi(x)$$

Find w
and the prediction function is:

$$f(x) = w \cdot \Phi(x) + b$$

Example: polynomial map

$$\Phi : R^2 \to R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)} x_1 x_2, x_2^2)$$



Linear

Support Vector Machines (SVM)

Problem: The map function could lead to a very high dimensionality, making it difficult to keep w in memory and solve the optimization problem.

Solution: dual form + kernel trick

Linear

Support Vector Machines (SVM)

Dual form. The optimization problem can be presented in its dual form, with Lagrangian multipliers

We try to find:

Find α_i, minimising: $\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i x_j$

subject to:

$$\sum_{i=1}^{m} y_i \alpha_i = 0$$

$$0 \le \alpha_i \le C, i = 1, \ldots, m$$

The classification function is now:

$$f(x) = \sum_{i}^{n} \alpha_i y_i x_i \cdot x + b$$

Moreover, there is a relation between w y α

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Linear

Support Vector Machines (SVM)

Dual form

Advantage: Both in the function to solve and the decision function data appear as dot products

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

$$f(x) = \sum_i \alpha_i y_i x_i \cdot x + b$$

For the non linear problem, we include a map $\boldsymbol{\Phi}$:

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \Phi(x_i) \cdot \Phi(x_j)$$

$$f(x) = \sum_i \alpha_i y_i \Phi(x_i) \cdot \Phi(x) + b$$

Linear

Support Vector Machines (SVM)

Kernel trick

Lets consider the function K (kernel):

$$K(x_i, x) = \Phi(x_i) \cdot \Phi(x)$$

If K is the dot product, we maintain the problem in the same input space, but by varying K we could get something like:

Map + dot product (but without making an explicit mapping!)

Example:

Polynomial kernel:

$$K(x, x') = (x \cdot x')^2$$

Which is equivalent to the previous mapping:

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3 \quad (x_1, x_2) \to (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

Linear

Support Vector Machines (SVM)

Kernel trick:

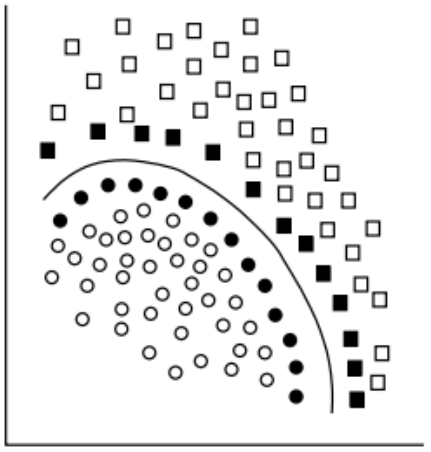
Other kernels.

Polynomial

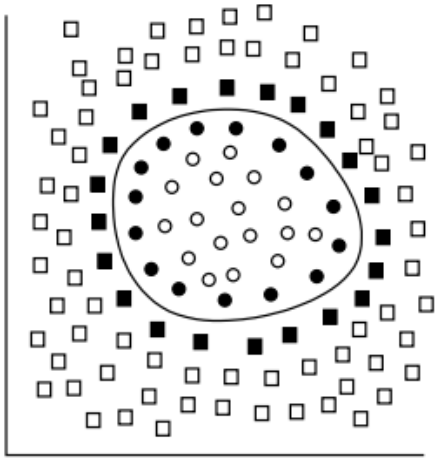
$$K(x, x') = (x \cdot x' + c)^q$$

RBF (adds a small "hill" on each point)

$$K(x, x') = e^{-\gamma \|x_i - x\|^2}$$



polinomial

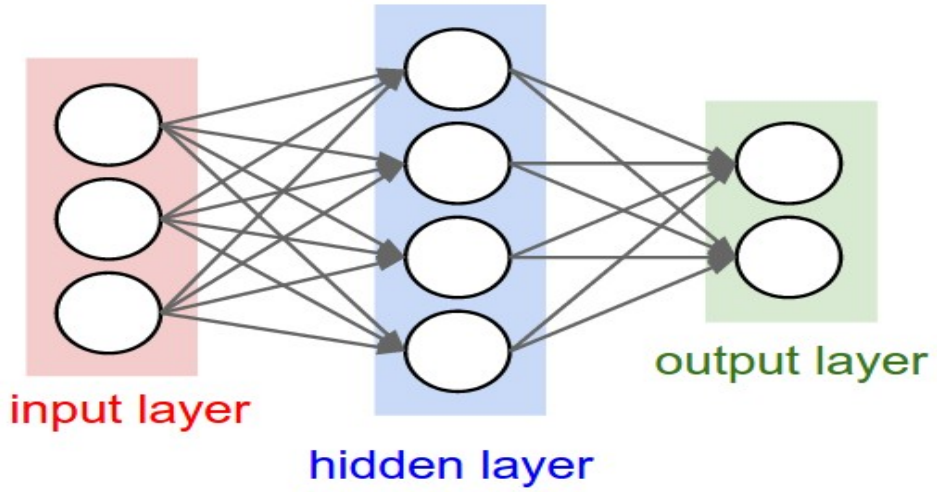


RBF

Linear

Neural Networks

- A neural network can be seen as a complex linear classifier able to learn non-linear, multi-class datasets
- It also uses gradient descent for training
- Basic architecture
  - Neurons disposed in layers
  - The input layer receives a sample
    - For example: each gene expression value
  - The hidden layer(s) combine(s) the output from previous layers
  - The output layer gives predictions
    - For example: one output neuron gives the score for each possible class (patient possible conditions)



Linear

Neural Networks

Each neuron is composed with a set of weights w, a bias b and an **activation function** f (a.ka. Non-linearity)

Neuron output (known as "activation"):

$$a(x)=f\left(\sum_i w_i x_i+b\right)$$

Possible activation functions f:

Sigmoid

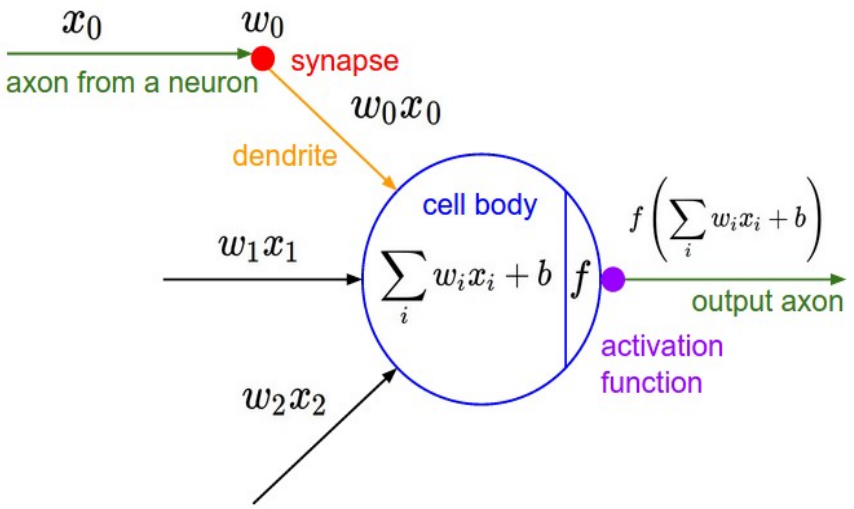
$$\sigma(x)=\frac{1}{(1+e^{-x})}$$

Tanh

$$\tanh(x)=2\,\sigma(2x)-1$$

ReLU (Rectified Linear Unit)

$$f(x)=max(0,x)$$



Linear

Neural Networks

Training. Backpropagation algorithm

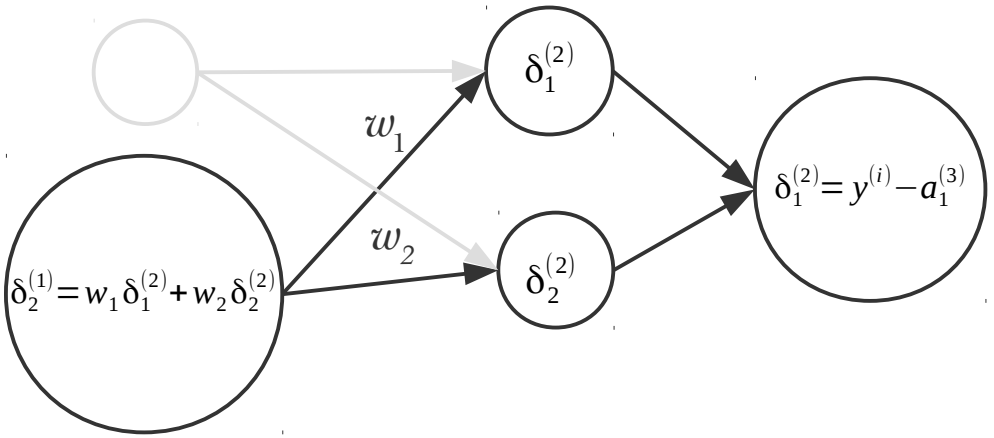
It is a method to compute the partial derivative (δ) at each neuron in order to update its weights with gradient descent (as in previous methods)

δ is calculated from the last layer down to the first

In the last layer, the gradient depends only on the expected output

In intermediate layers, the gradient depends on the gradient of the next connected neurons, weighted by the connection's weights

Intuition:



Linear

Neural Networks

Training. Backpropagation algorithm

Once δ_n is calculated for each neuron n, and given that (which is proven):

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \delta_n^{(L)} a_n$$

We can apply gradient descent to update weights a learning rate α

$$w_{ij} := w_{ij} - \alpha \delta_n^{(L)} a_n$$

Linear

Convolutional Neural Networks (CNN)

Deep Learning: subfield of Machine Learning able to learn from datasets without doing *feature engineering*

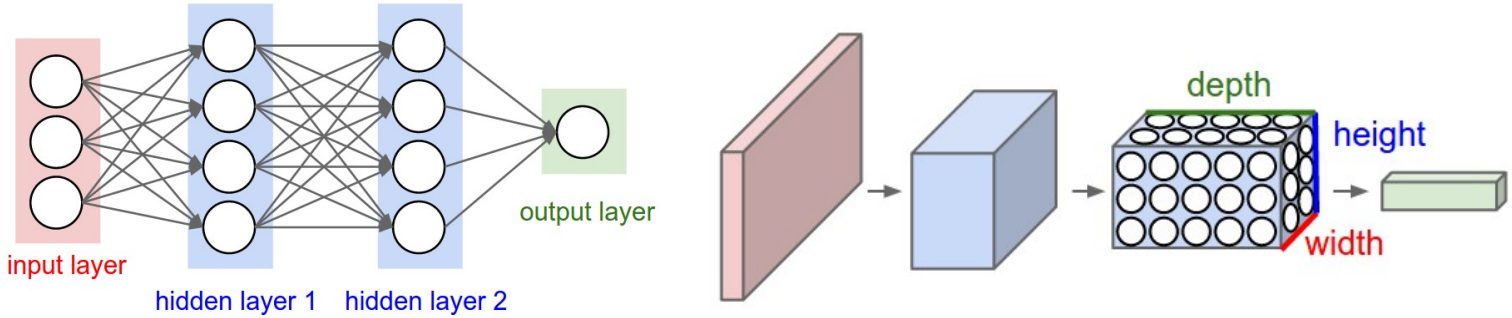
For example, in order to detect objects in an image, a preprocessing step trying to find basic shapes (features) focusing on the specific task was explicitly programmed. Deep Learning techniques try to avoid this step, working with raw data directly.

Convolutional Neural Network (CNN): A special type of *deep* neural network intended for image processing.

Linear

CNN vs classic Neural Networks. Key differences:

- In CNN, layers are *volumes* of neurons having width, height and depth.
- Neurons are not *fully connected* to the previous layer, because it will not scale well for images (huge number of connections).
  - They only process a square portion of the input.
- Weights are *shared*. Neurons at the same depth in the same volume *share* they weights.
  - It is reasonable for image processing as well as it reduces dramatically the number of parameters.



Linear

Convolutional Neural Networks (CNN)

Types of layers in a CNN:

Convolutional Layer.

The most important layer. Neurons are connected to portions of the input volume.

RELU layer.

Applies an elementwise activation, such as max(0, x). The input volume's dimensions are not changed

POOL layer.

Reduces the the spatial dimensions (width and height).

Fully connected layer.

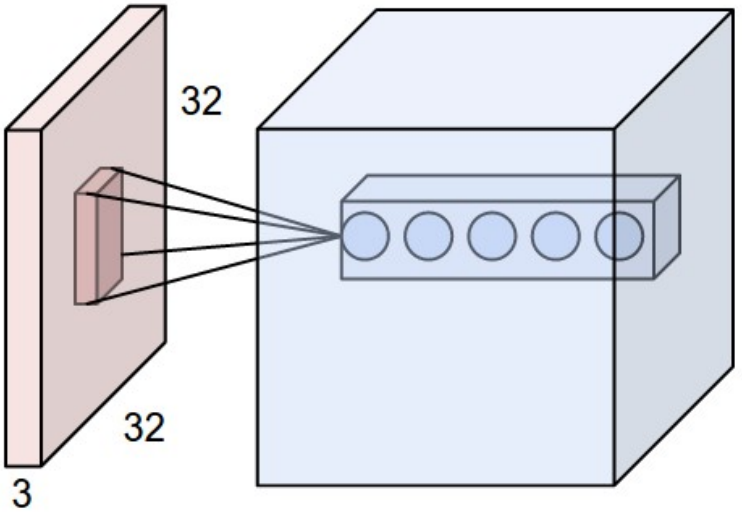
Typical final layer, which is fully connected to the input and has a neuron per desired output (for example, 10 neurons for a 10-class classification problem)

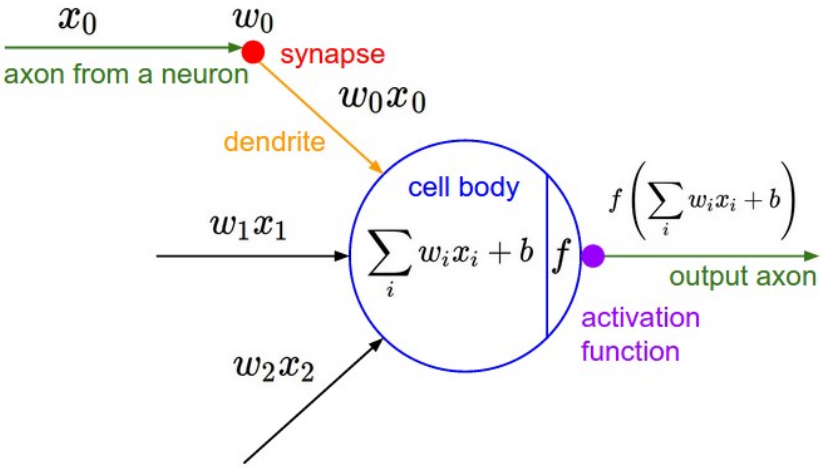
Linear

Convolutional Neural Networks (CNN)

Convolutional Layer

- Each neuron receives a small portion of the input's width and height (but all the input's depth)
- Neurons in at the same depth in the conv. layer share they weights.
- Each set of neurons at the same depth is called *a filter*
  - They process the input image applying the same operation in a sliding window (convolution), so they are like applying a filter to the image

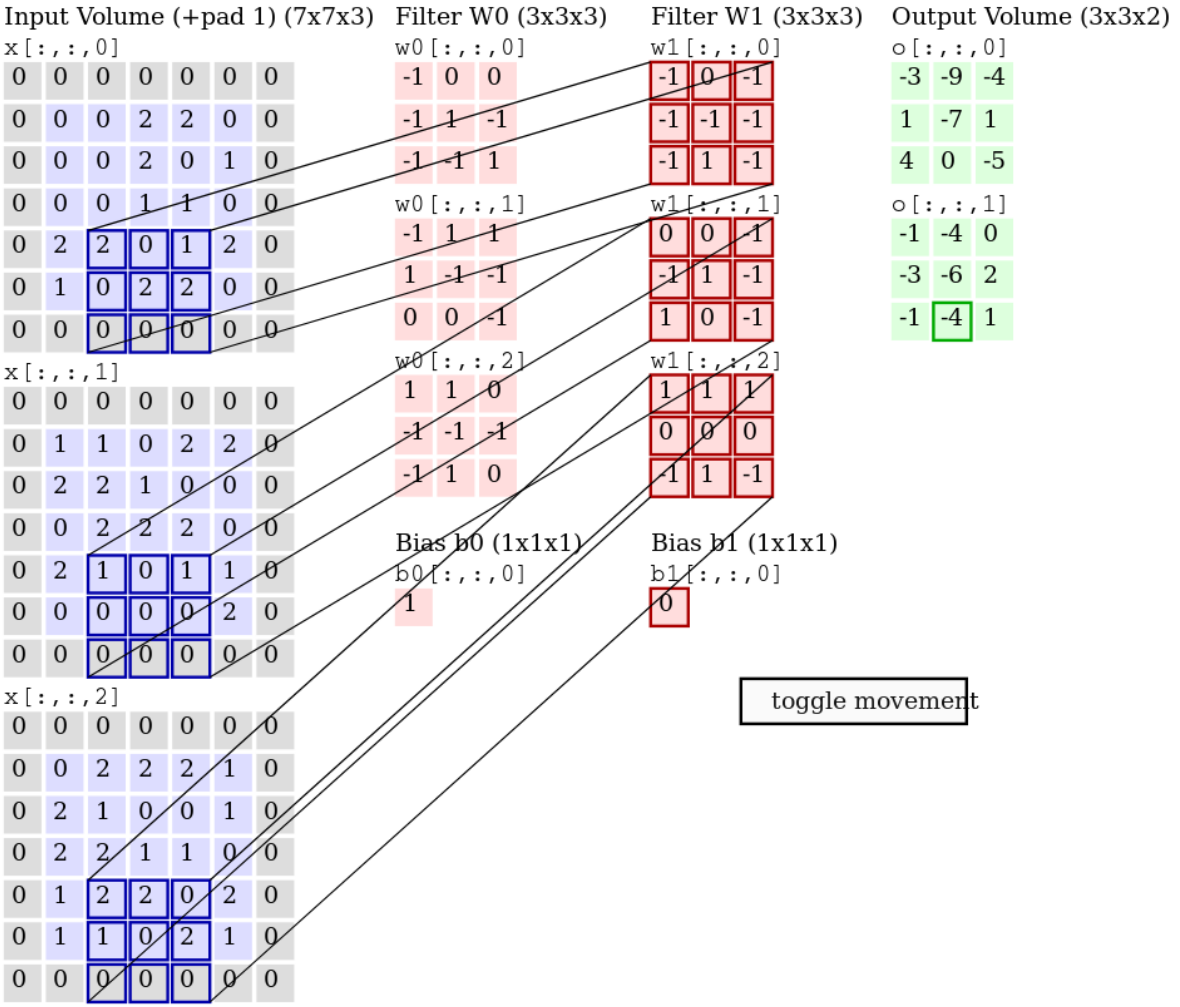




Linear

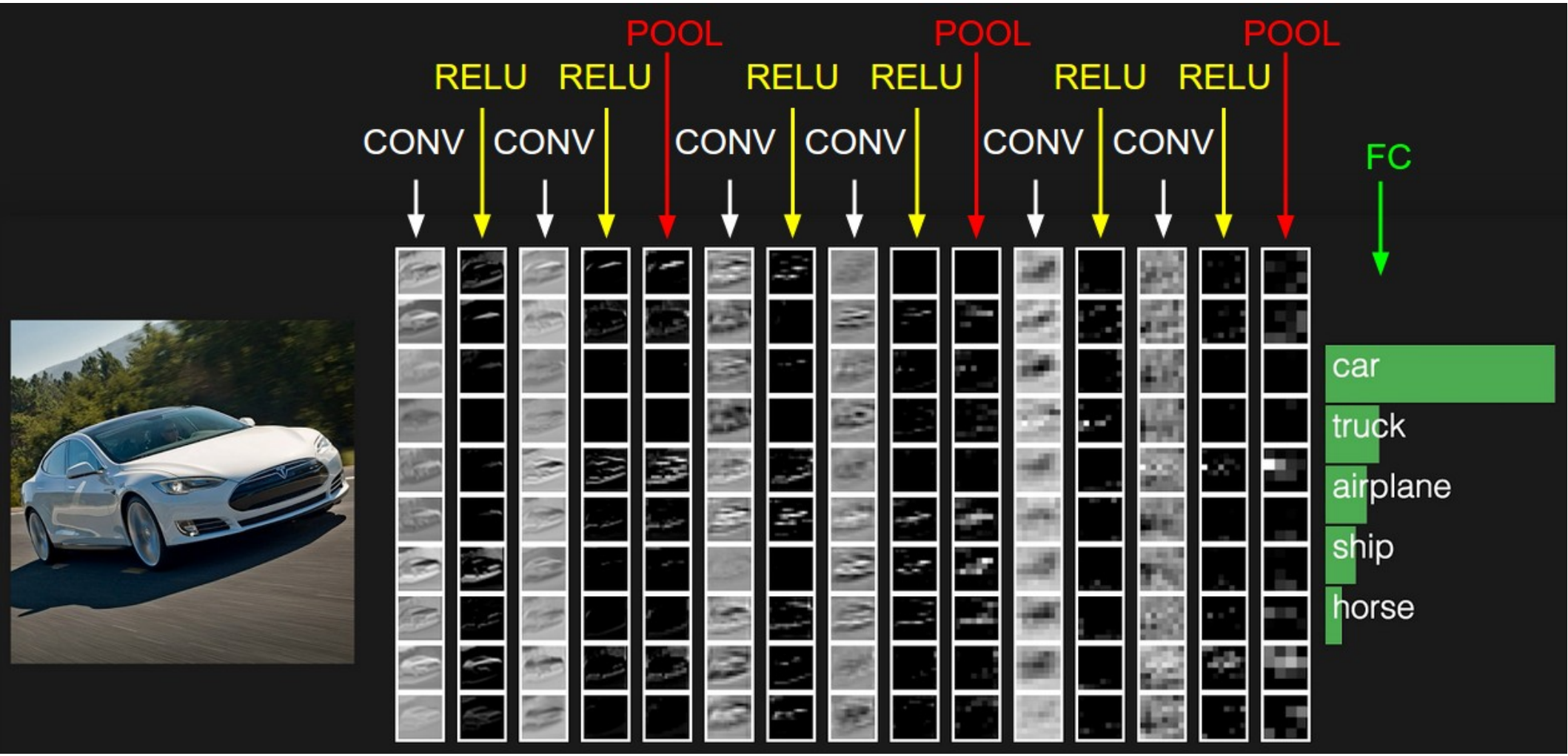
Convolutional Neural Networks (CNN)

Example of convolutional layer values



Linear

Convolutional Neural Networks (CNN)



Classifier ensembles

Employ several classifiers in order to improve performance

Ensemble vs. other classifier combination approaches:

Ensembles employ many classifiers (of one or several types). Moreover, its number usually varies depending on the training data

The combination of a reduced number of classifiers is not considered an ensemble (less than 10), specially when they are intended to integrate data from heterogeneous sources

Motivation:

Improve the statistical performance

Probably, the performance of the classifier combination is lower than a specific classifier, but the risk of choosing a bad classifier is reduced

Computational aspects

Usually it is unfeasible to try all classifiers to see which is the best one. By combining several types, the risk is reduced

Representation capabilities: divide and conquer

Making each classifier to focus on a subpart of the input data space, **results can be improved**

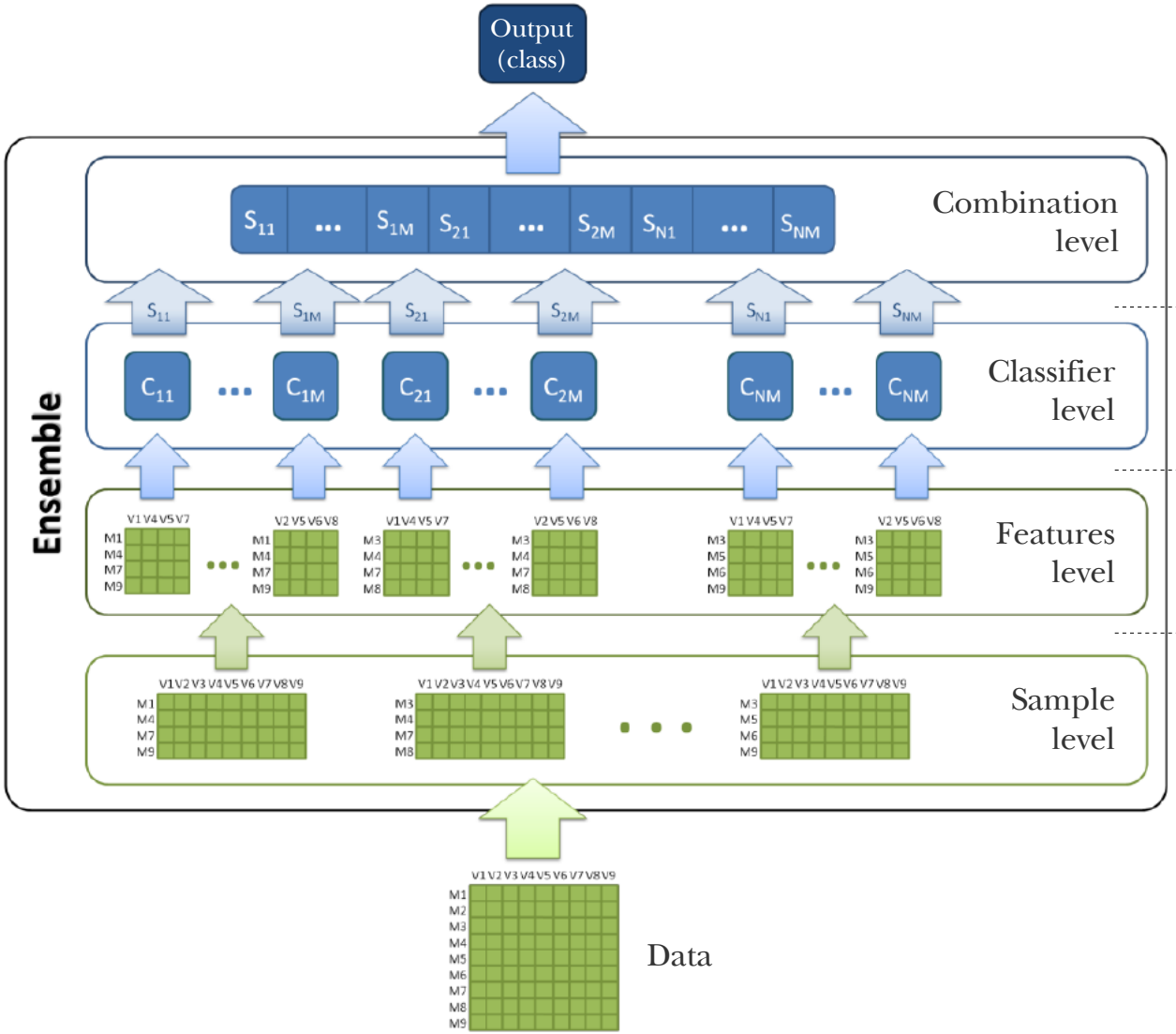
Advantages regarding data volume

Very high data volume. Scalability, due to the possibility of train several classifiers per data subspace concurrently

Very low data volume. Resampling data and train a classifier on each sample

Classifier ensembles

Ensembles general framework



Output type:
(I) class, (II) ranking p.class, (III) p. class confidence.
Distinct methods for each type

Homogeneous vs. Heterogeneous
Construction: Independent vs. Dependent
Classification: Parallel vs. Sequential vs. div. and conq.

Natural grouping (problem dependent)
Random selection
Not random (favourite class, iterative, incremental)
By optimization (opt. diversity and performance)

Sampling methods (with replacement)
Weights usage
Partition methods (without replacement)
New sample creation methods

Classifier ensembles

Diversity

Central and desired feature in classifier ensembles

If we find a set of classifiers diverse enough, that do not make the same errors, then we can improve the performance of the base classifiers

How to achieve diversity:

- Modify classifier parameters
- Modify input training data (sampling, partition, creation).
- Divide search space (divide and conquer)
- Mix several types of classifiers

Classifier ensembles

Diversity

Measuring diversity

For each pair of classifiers:

Taking this matrix:

	C_j hit (1)	C_j fail (0)
C_i hit (1)	N^{11}	N^{10}
C_i fail (0)	N^{01}	N^{00}

$$Q_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}}$$

Q statistic [-1, 1].
0 = independent

$$\rho_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11}+N^{10})(N^{01}+N^{00})(N^{11}+N^{01})(N^{10}+N^{00})}}$$

ρ correlation coefficient
Similar to Q

$$Des_{i,j} = \frac{N^{01}+N^{10}}{N}$$

Disagreement score

$$DF_{i,j} = \frac{N^{00}}{N}$$

Doble fault

Joint diversity of two classifiers

$$E = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{C - \lceil C/2 \rceil}\min\{F_i, (C - F_i)\}$$

Entropy

$$KW = \frac{1}{NT^2}\sum_{i=1}^{N} F_i \cdot (C - F_i)$$

Kohavi-Wolpert Variance (T=number of classifiers)

(F_i number of classifiers failing i sample).

Classifier ensembles

Some examples

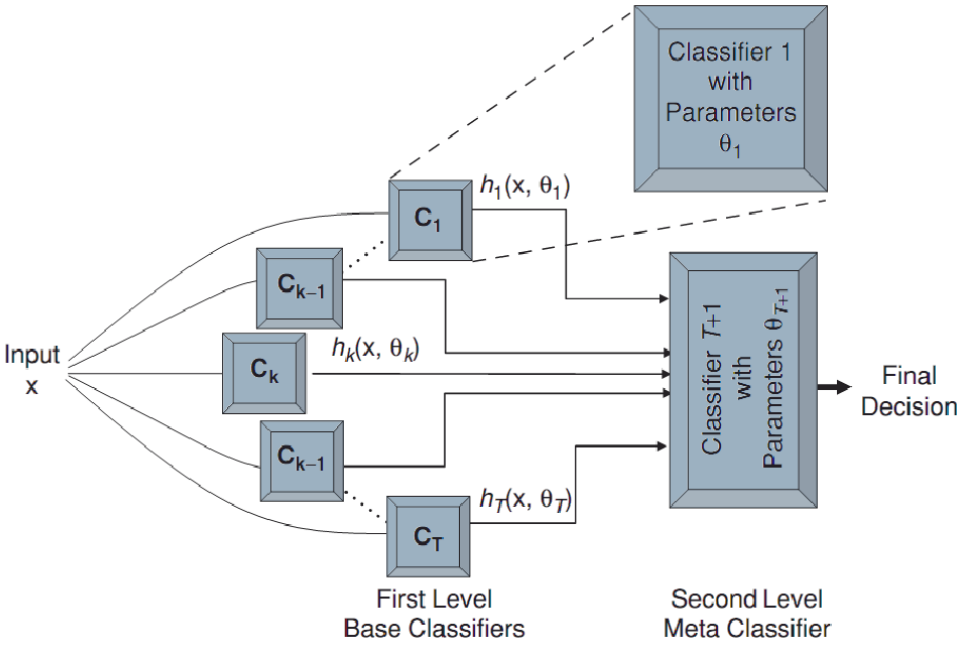
- Bagging (Bootstrap AGGregatING)
  - Several classifiers over subsamples by using resampling with replacement
  - Combining by majority vote
- Random Forest
  - Bagging variation
  - Builds multiple decision trees by varying:
    - Parameters randomly
    - Taking a subset of samples and/or attributes randomly
- AdaBoost
  - Sequential training of several classifiers of the same type and parameters
  - On each iteration failing samples of the previous iteration have more weight
  - Combination by performance-weighted voting

Classifier ensembles

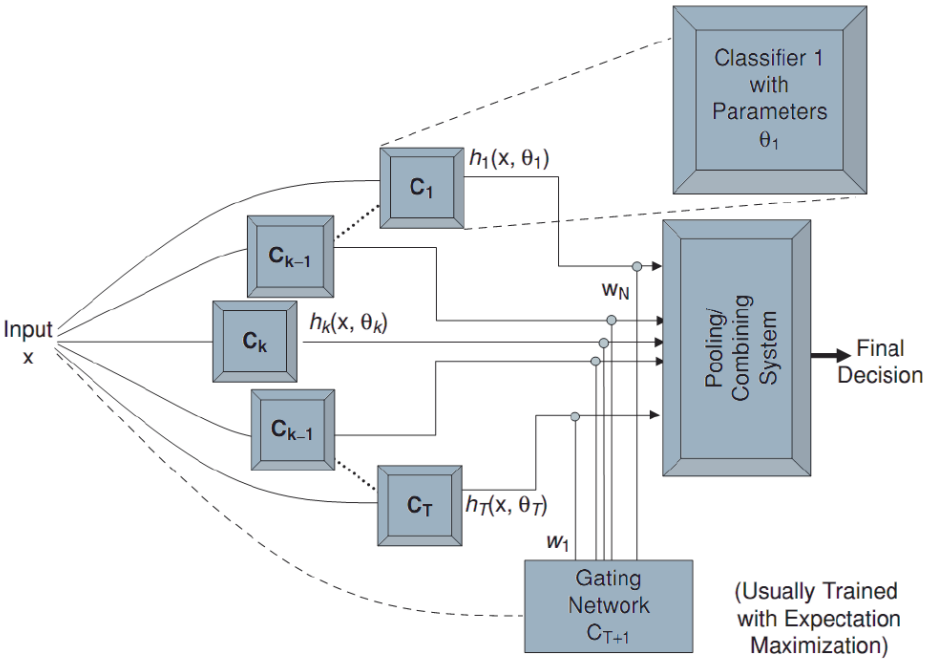
Some examples

Non-ensemble approaches

Stacking
A classifier learns how to generate a classification taking as input the output of base classifiers



Mixture of experts
A neural network learns which classifiers should vote given input data



References

Witten I.H. & Frank E (2005) Data Mining: Practical Machine Learning Tools And Techniques. Morgan Kaufmann. 2nd Edition.

Kuncheva, L.I. (2004). Combinining Pattern Classifiers: Methods and Algorithms. Wiley-Intersciente. 1st Edition.

Convoluational Neural Networks for Visual Recognition
http://cs231n.github.io

Wikipedia

Machine Learning

Daniel Glez-Peña &
Hugo López-Fernández
@SINGgroup
www.sing-group.org