A Quick Introduction to Machine Learning and How to evaluate the models

---

Dino Ienco
dino.ienco@irstea.fr
Researcher @IRSTEA

Outline

Introduction to ML

Supervised Learning:
- + Random Forest
- + SVM
- + Multi-layer Perceptron

Model Evaluation:
- + Confusion Matrix
- + Evaluation Metrics
- + Training / Validation / Test
- + Hold-Out / Cross-Validation / Leave One Out

Introduction

In many real world domains we need decisions:

- Decide if a mail is SPAM or not
- Decide if a medical patient is sick or not
- Decide if a people can be interested to a product or not
- Decide if a web page speaks about politic or sport
- Decide if bank transaction is fraudulent or not

Introduction

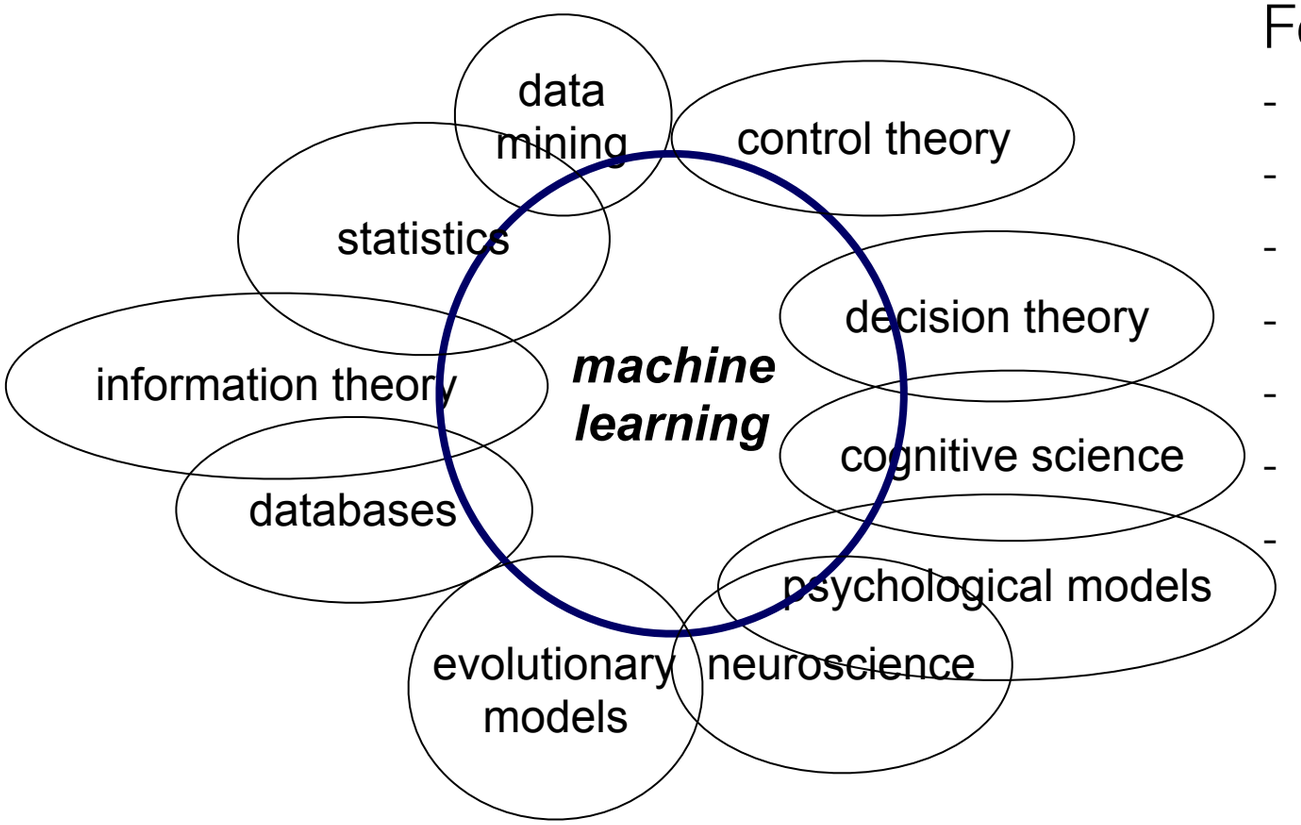
- Each decision requires time to be taken
- Each decision can also be expensive to take

- Nowadays data sources produce Huge quantity of data :
  - News feeds to be categorized and archived
  - Predict opinion from tweets data
  - Monitoring sensor data (such as remote sensing images)

Introduction

Machine Learning:

Build a model in order to automatically and autonomously make a decision



For Example, make a decision about:

- The category of images
- Land Cover of Hyperspectral pixels
- Classify gene species
- Categorize text document
- Land use from satellite images Time series
- If a mail is a Junk mail or not
- etc…

Introduction

Supervised Learning:
 - needs **Data** and **Labels** in order to train a model to perform classification for unseen data

Unsupervised Learning:
 - needs **Data** and **NO Labels** in order to learn model a to describe the data

Semi-supervised Learning:
 - needs **Data** and **some Labels** in order to train a model to perform classification for unseen data

Supervised Learning

Formally, in a Supervised Learning (SL) scenario we have:

Training Data: $\{(X_i, Y_i)\}_{i=1}^{n}$

Testing Data: $\{(X_j, ?)\}_{j=1}^{m}$

Where, **training** and **test** data are **disjoints**.

We name **Y target** or **class variable**

GOAL: **learn a function f(.)** from the training data in order **to predict/ estimate the target variable** on the test data

When the **Class Variable is**:

- **Discrete** (it can takes a set of discrete values: i.e. a set of land cover classes) we talk of **Classification** (Binary or MultiClass)

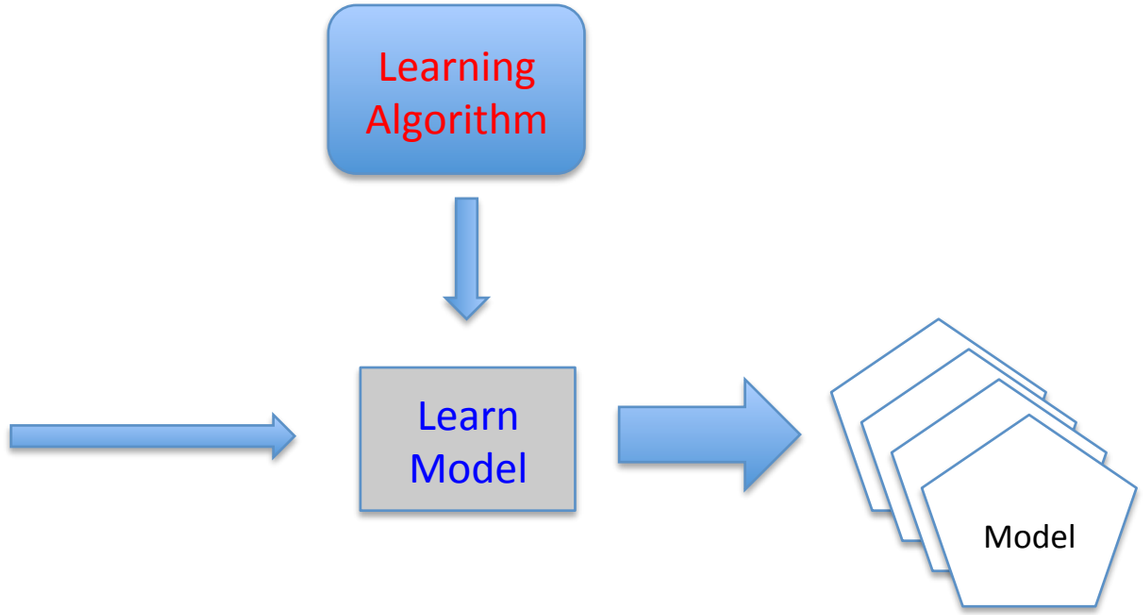
- **Numerical** (i.e. a physic/biology parameter) we talk about **Regression**

Supervised Learning

Classification Task

Training Data

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

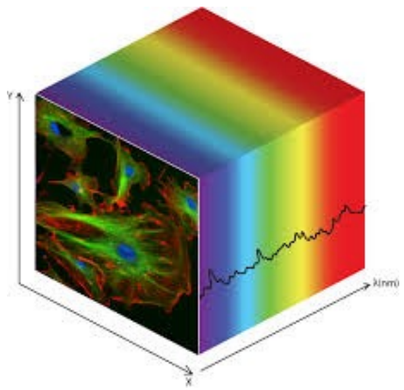


Test Data

Tid	Refund	Marital Status	Taxable Income	Cheat
1	No	Married	80K	?
2	No	Single	100K	?
3	Yes	Single	90K	?
4	No	Married	120K	?
5	Yes	Divorced	130K	?

Supervised Learning (ex. Hyper spectral data)

Supervised Learning and Hyperspectral Data



In the case of hyper spectral data:

Training data: Pixels / Objects / Images
with associated Label Information (Ground Truth - i.e. Land Cover)

Testing Data: Pixels / Objects / Images
for which we do not have the class labels and we want to determine it

Supervised Learning

Many different classification algorithms:
- Decision Trees, Support Vector Machine (SVM), Random Forest, Naive Bayes, Multi-layer perceptron

Each kind of approach has different hypothesis:

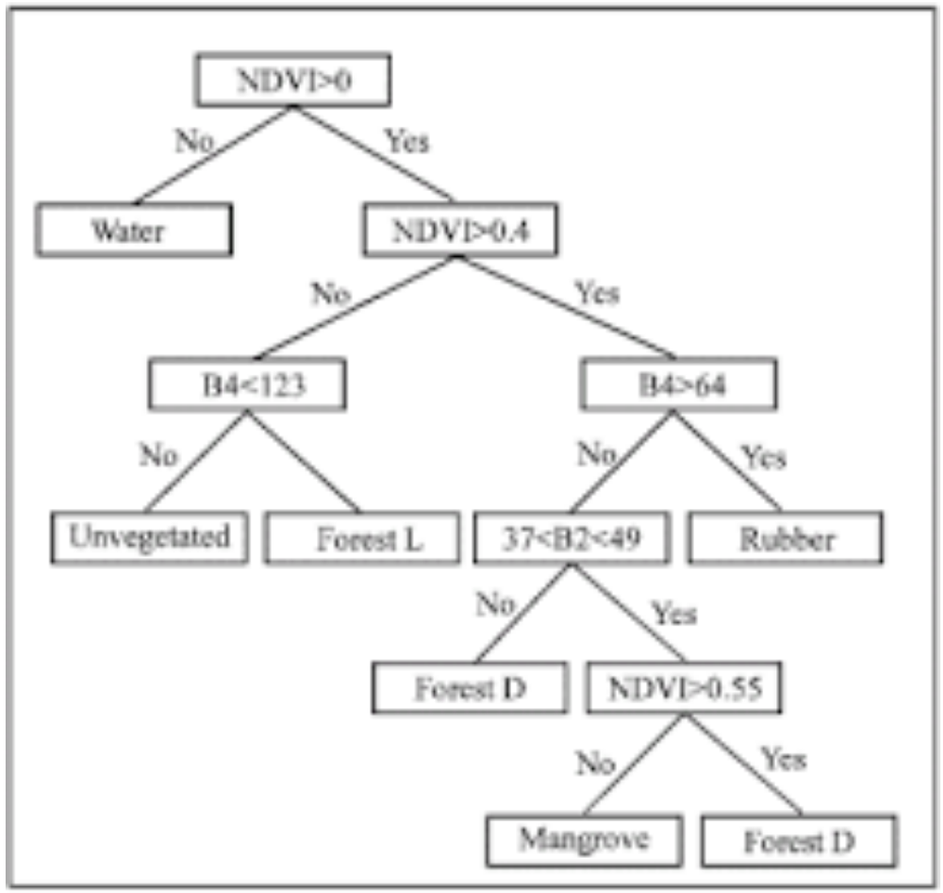
- **Decision Trees**: decision are done thresholding the attributes value
- **SVM**: data are linearly separable in some inducted space
- Naive Bayes: data attribute are independent each others
- **Multi-layer perceptron**: multiple non linear combination can separate classes

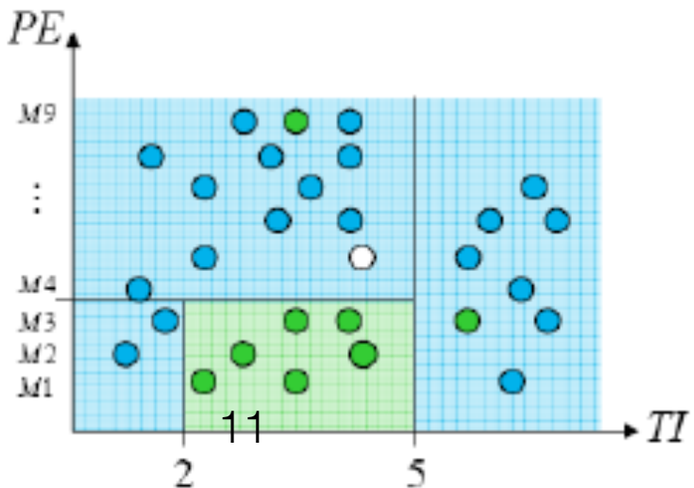
Each method has its advantages and disadvantages
No method fits all the cases or datasets

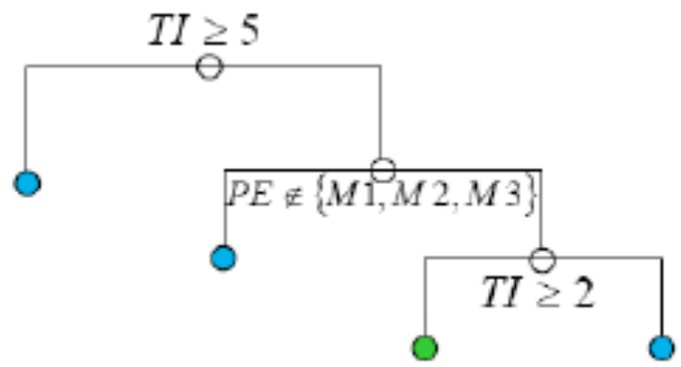
Random Forest

Decision Tree:

- Inductive Algorithm that **Greedy** and **Recursively partitions** the data space
- **Splits are learnt** on training data X
- Perform **Test** on **data Features**
- **Leaves** of the Tree contains the **class** to Predict
- A new example is classified **following a path in the tree**
- It can use **a subset of the features** to build the classification model





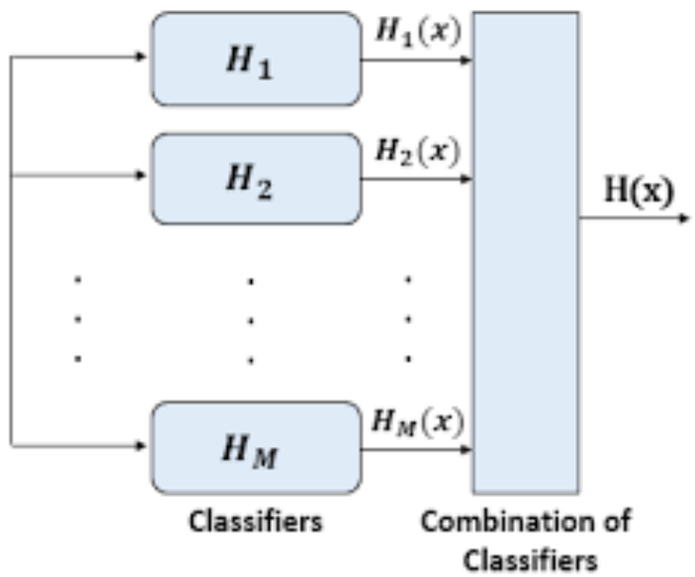


Random Forest

- **Random forest** (or **random forests**) is an **ensemble** classifier that consists of many (random) decision trees and outputs the class that is the mode of the class's output by individual trees.

- The method combines Breiman's "**bagging**" idea and the random selection of features.

Ensemble: Instead of use a single classifier, use a set of classifiers and combine their decisions to supply the final classification.



Bagging: Sampling portion of the dataset (w or w\o replacement) and train one classifier per subsample.

Random Decision Tree: Select the features for the split at random

Random Forest: To Summarise

Pros:

- Empirically, for many data sets, it produces a highly accurate classifier.
- It runs efficiently on large databases considering both training and test time.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.

Cons:

- Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.

Implementation availables: Matlab, R, Python, Java, C, C++, Ruby

Recent Tree-based Ensemble classifiers:

- Gradient Boosting Trees
- Extreme Gradient Boosting Trees

Support Vector Machine (SVM)

Kernel Trick: Map data to *higher-dimensional space* where they will be *linearly separable*.

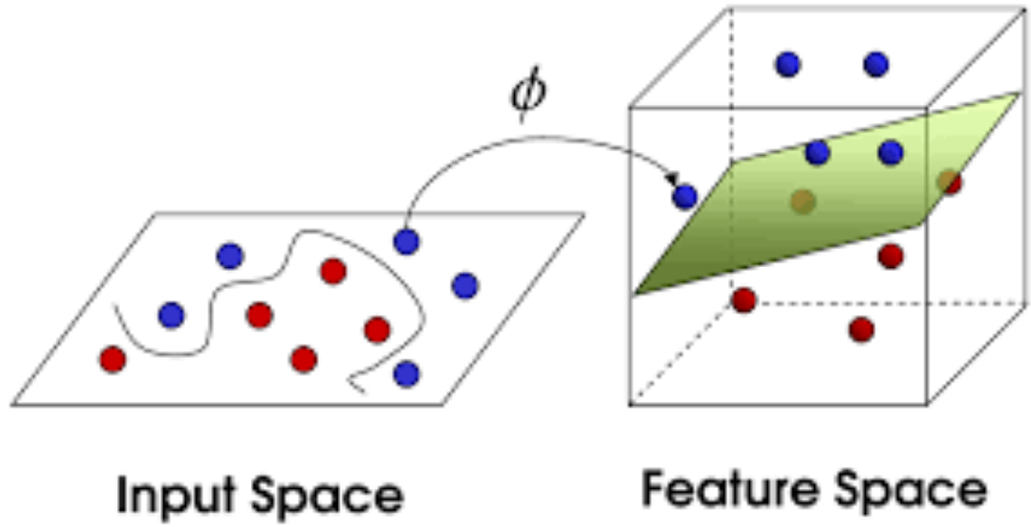
Learning a Classifier

- Optimal linear separator is one that has the *largest margin* between positive examples on one side and negative examples on the other
- *= quadratic programming optimization*

$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m$$

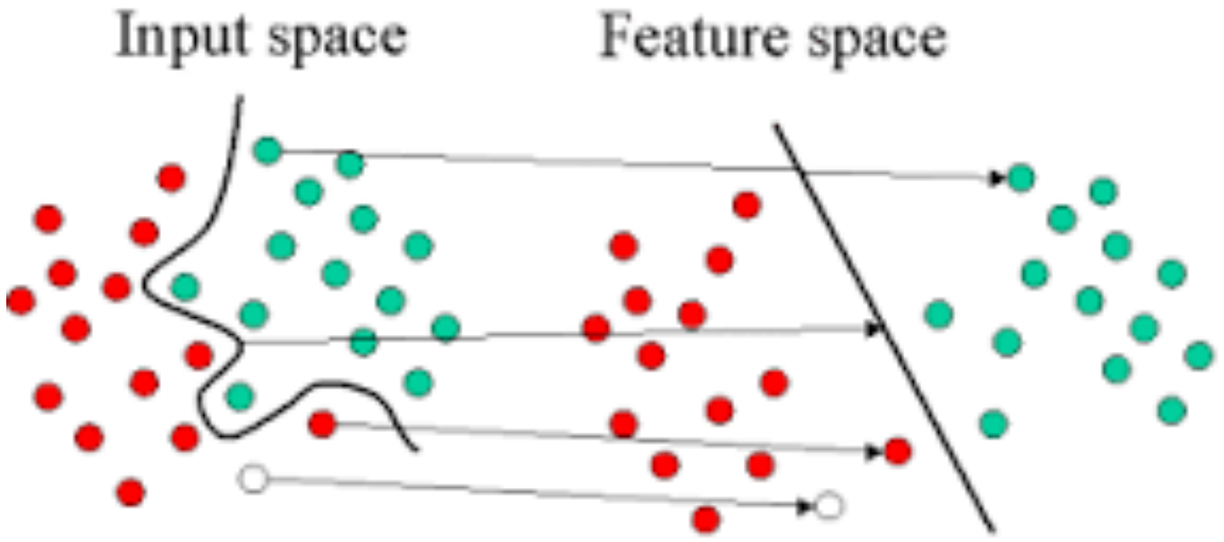
$$\xi_i \geq 0, \quad i = 1, \ldots, m.$$



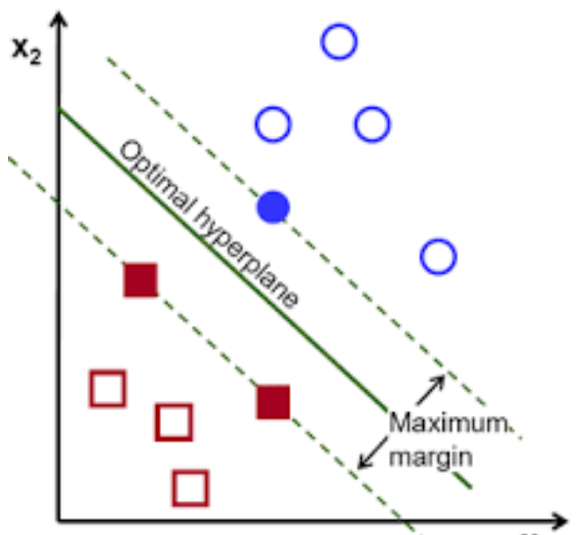
Support Vector Machine (SVM)

Key Concept: Training data enters optimization problem in the form of *dot products* of pairs of *points*.

- ***Kernel function*** **$K(x_i,x_j)$**
  - Function that can be applied to pairs of points to evaluate dot products in the corresponding (higher-dimensional) feature space F (*without having to directly compute* F(x) *first*)
  - Standard Kernel: Linear, Polynomial, **Radial Basis Function** (RBF)



- ***Support vectors***
  - Weights associated with data points are *zero* except for those points nearest the separator (i.e., the *support vectors*)



Support Vector Machine (SVM)

SVM: To Summarise

Pros:

- Empirically, for many data sets, it produces a highly accurate classifier.
- It is robust to noisy data.
- It is useful for both Linearly Seperable (hard margin) and Non-linearly Seperable (soft margin) data.
- Thanks to the 'Kernel Trick' the feature mapping is implicitly carried out via simple dot products

Cons:

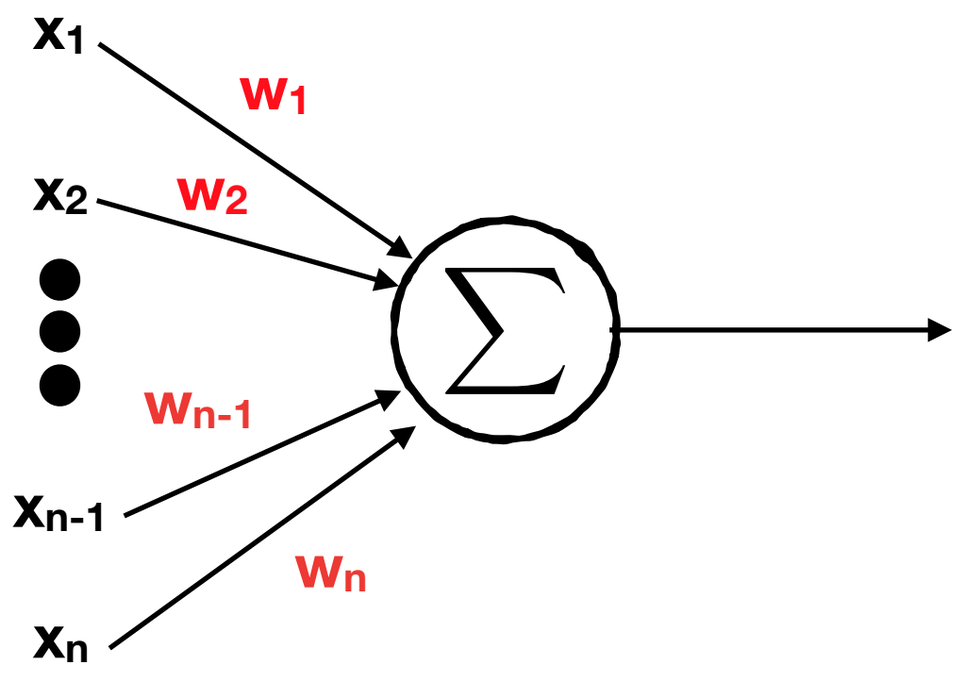
- Depending on the implementation, it has problems to handle big volume of data (scalability issues)
- Estimate the correct parameters (i.e. C and Sigma for RBF kernel) can take times

Implementation availables: Matlab, R, Python, Java, C, C++, Ruby

Multi-Layer Perceptron (MLP)

Perceptron

- First neural network learning model in the 1960's
- Simple and limited (single layer models)
- Basic concepts are similar for multi-layer models so this is a good learning tool
- Still used in many current applications (modems, etc.)



$$1 \quad \text{if} \quad \sum x_i w_i + b > 0$$

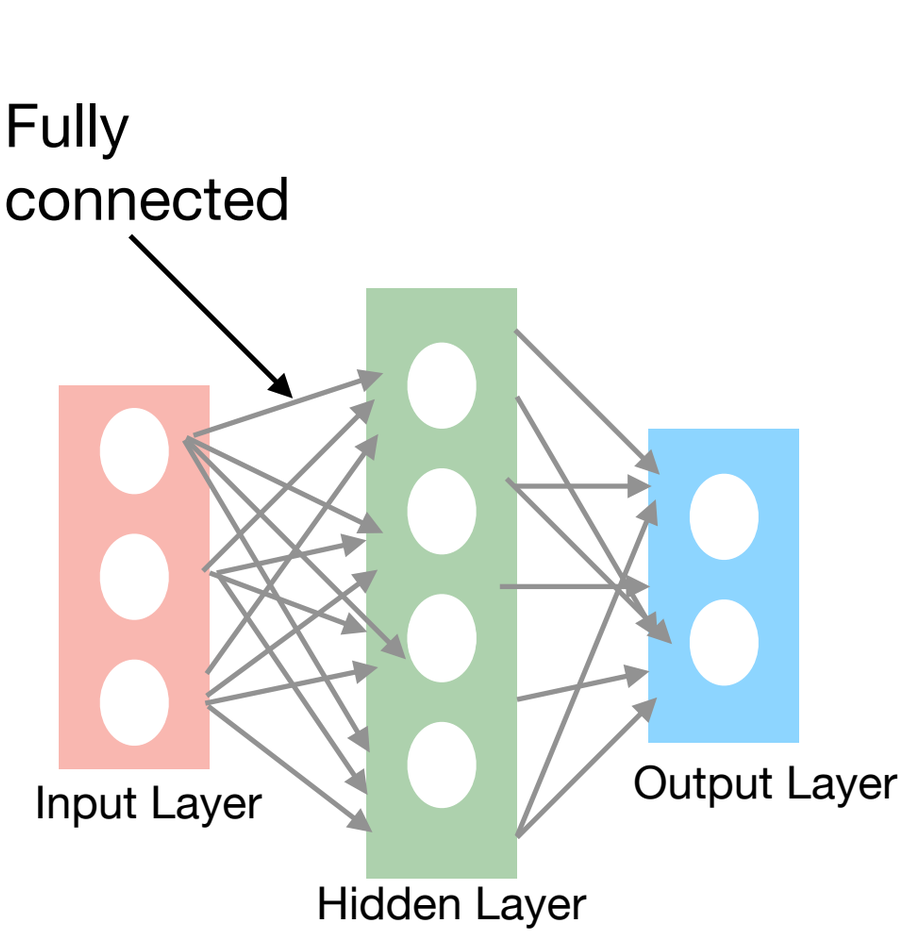
$$0 \quad \text{Otherwise}$$

- Learn weights such that an objective function is maximised.

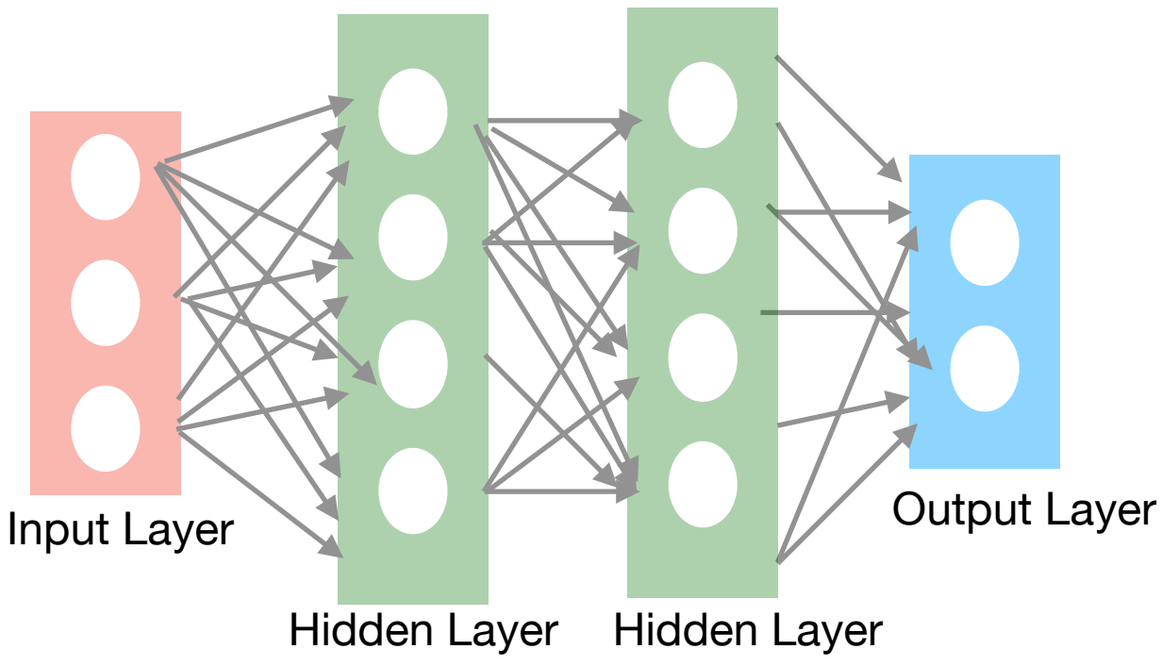
Perceptron is a linear classifier Y= W x

Multi-Layer Perceptron (MLP)

Feed-Forward architecture
We have several layers (usually two hiddens) with multiple perceptrons
The activation function (the function computed by a perceptron) is not the linear one



2 Layers Neural Network



3 Layers Neural Network

Multi-Layer Perceptron (MLP)

MLP: To Summarise

Pros:

- Empirically, for many data sets, it produces a highly accurate classifier.
- It is robust to noisy data.
- It is useful for both Linearly Seperable and Non-linearly Seperable data.
- Nowadays, Neural Networks extensions (i.e. CNN, RNN, etc..) are state of the art classifier in several domains (Image, Text, Graph, etc…)

Cons:

- Many parameters to Tune and many possible variants
- Not so adequate to learn model from small training dataset
- Black-box model, explanation is still unclear

http://playground.tensorflow.org/

Implementation availables: Matlab, R, Python, Java, C, C++, Ruby, etc..

Remote Sensing Classification

Object-Based Analysis (OBIA):

i) segment the image to obtain Segment
ii) model is trained on segments/objects and test on segments/objects

Pixel-Based Analysis (PBIA):

i) the unit of analysis is the pixel.
ii) model is trained on pixel and test on pixels

Fixing the image to analyse:

OBIA:

i) Pros: Scale-up to bigger area (less objects to classify)
ii) Cons: Segmentation step before (parameters, choice of the algorithm…)

PBIA:

i) Pros: We use directly the image information
ii) Cons: Resource-Intensive (more samples in the training and test set)

Model Evaluation

Confusion Matrix

Evaluation Metrics

Training / Validation / Test

Hold-Out / Cross-Validation / Leave One Out

Confusion Matrix

How to evaluate the Classifier's Generalization Performance?

- Assume that we test a classifier on some test set and we derive at the end the following *confusion matrix*:

		Predicted class		
		Pos	Neg	
Actual class	Pos	*TP*	*FN*	*P*
	Neg	*FP*	*TN*	*N*

Evaluation metrics

Metrics for Classifier's Evaluation

- Accuracy = $(TP+TN)/(P+N)$
- Error = $(FP+FN)/(P+N)$
- Precision = $TP/(TP+FP)$
- Recall/TP rate = TP/P
- FP Rate = FP/N

Actual class \ *Predicted class*	Pos	Neg	
Pos	*TP*	*FN*	*P*
Neg	*FP*	*TN*	*N*

$$FMeasure = \frac{Precision \times Recall}{Precsion + Recall}$$

Confusion Matrix and Metrics

Accuracy

		Predicted class	
		Pos	Neg
Actual class	Pos	*TP*	*FN*
	Neg	*FP*	*TN*

Accuracy = (TP + TN) / (TP+FN+FP+TN)

Confusion Matrix and Metrics

Issues with <u>Accuracy</u>?

If <u>Accuracy</u> is used:

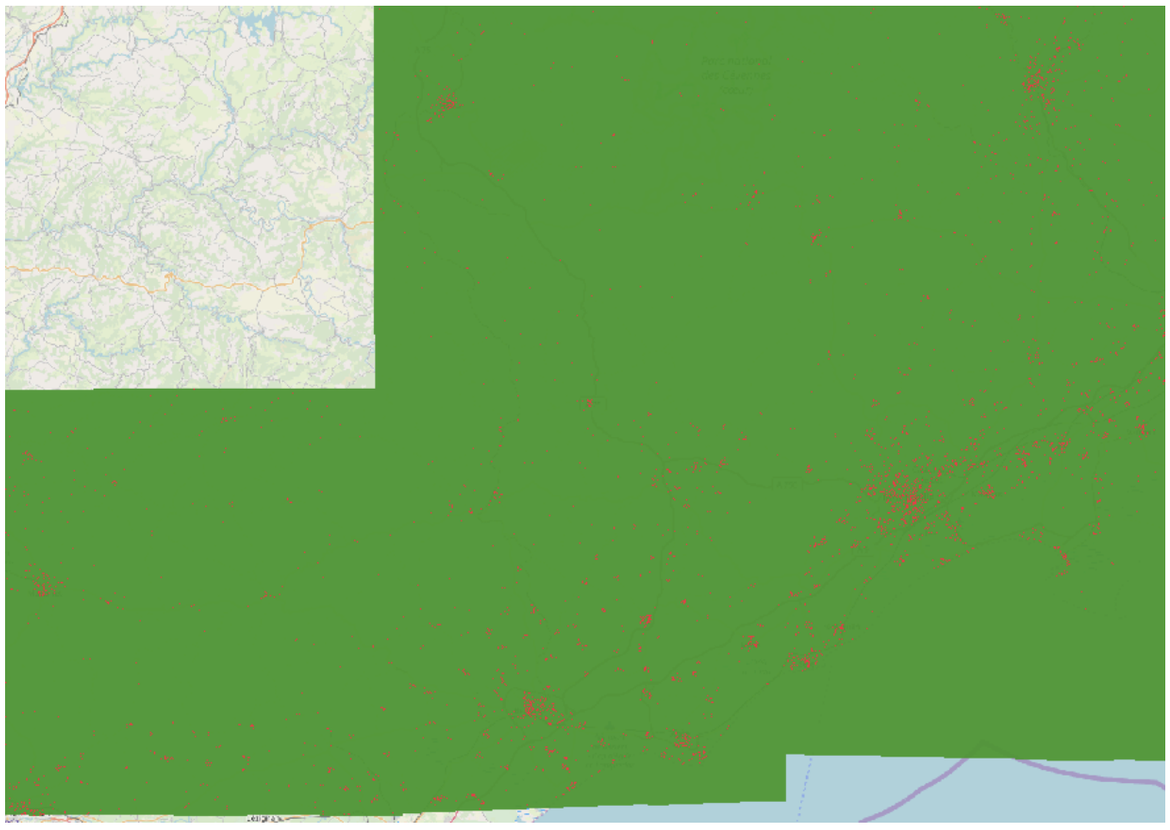
Example: in Urban Detection at large scale, there are many more non-urban zone than urban one

Presume only 1% of the zone is covered by Urban

Then:

Classifier that predicts <u>all the images</u> as Non-Urban would have 99% accuracy!

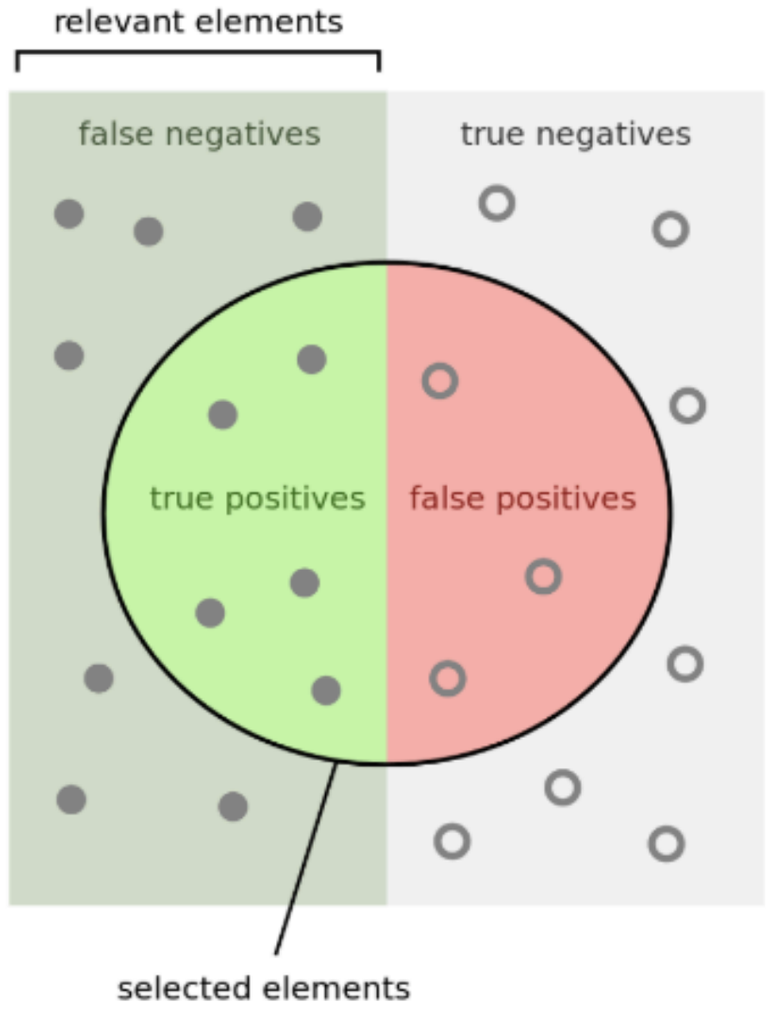
Seems great, but it's not…



Confusion Matrix and Metrics

Precision & Recall

		Predicted class	
		Pos	Neg
Actual class	Pos	*TP*	*FN*
	Neg	*FP*	*TN*

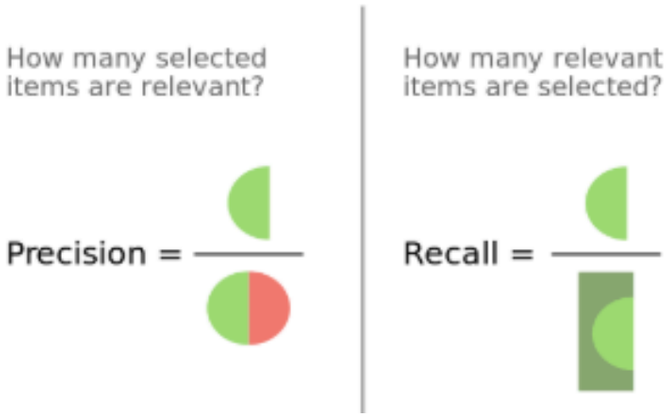


Precision: fraction of retrieved docs that are relevant = P(relevant|retrieved)

Recall: fraction of relevant docs that are retrieved = P(retrieved|relevant)

Precision P = tp/(tp + fp)

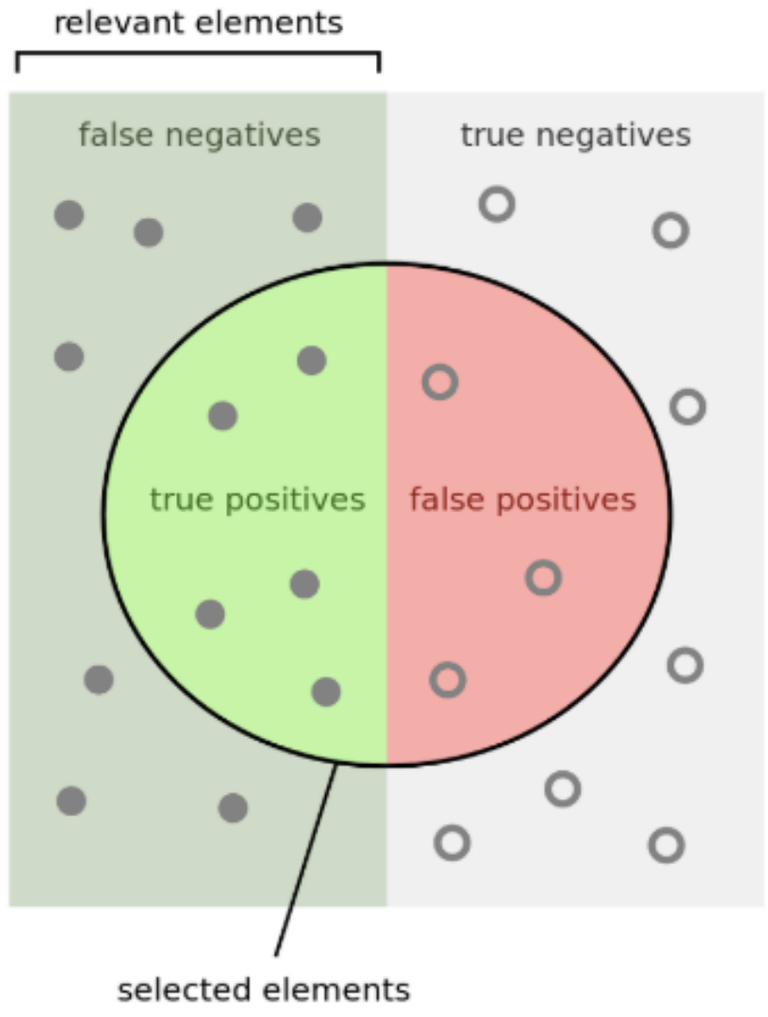
Recall R = tp/(tp + fn)



Confusion Matrix and Metrics

Precision & Recall

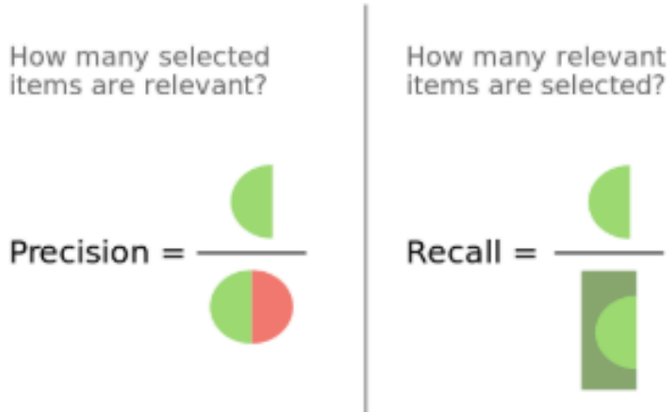
Widely used metrics when successful detection of one class is considered more significant than detection of the other classes



Interesting when we have unbalanced scenario in which classes have different distribution

$$\text{Precision}, p = \frac{TP}{TP + FP}$$

$$\text{Recall}, r = \frac{TP}{TP + FN}$$



Confusion Matrix and Metrics

- F-measure: combines precision and recall into a single metric, using the *harmonic mean*
  - *Harmonic Mean* of two numbers tends to be closer to the smaller of two numbers…
  - …so the only way F_1 is high is for both precision and recall to be high.

$$F_1 = \frac{2rp}{r+p} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

Harmonic mean is a conservative average

Kappa Measure

p_O: Observed probability

p_e: Estimated (Hypothetical) probability of chance probability

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e}$$

Kappa **is not an absolute value** as Accuracy, F-Measure, etc…

Kappa **estimates how much smarter** is a classifier (not how it works well on the considered data)

Kappa **estimates how far is** the model w.r.t. a random classifier with the same marginal probability (how much is far away from the chance)

Wikipedia

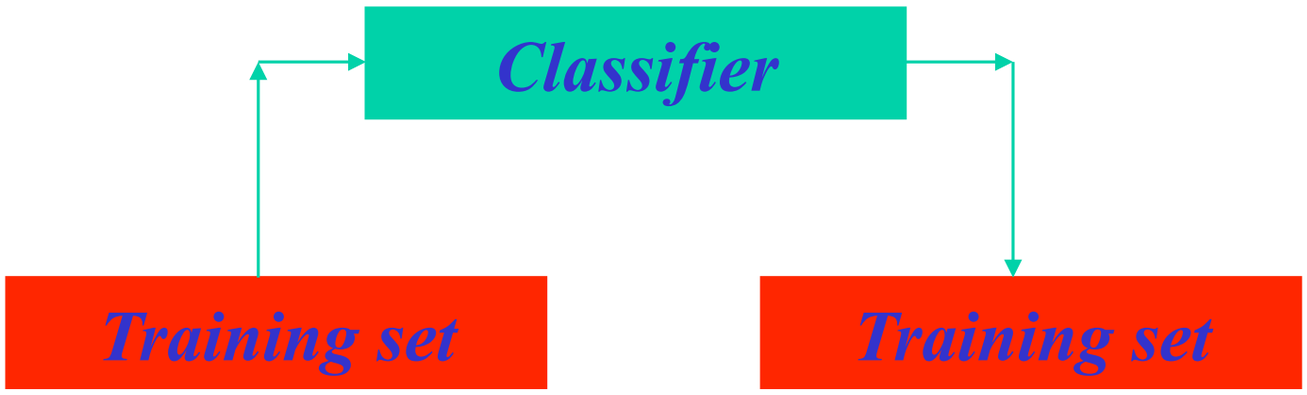
How to Estimate Metrics

We can use:

Training data;
Independent test data;
Hold-out method;
***k*-fold cross-validation method;**
Leave-one-out method;
Bootstrap method;
And many more…

Estimation with Training Data

- The accuracy/error estimates on the training data are *not* good indicators of performance on future data.



  - **Q: Why?**
  - **A:** Because new data will probably not be **exactly** the same as the training data!
- The accuracy/error estimates on the training data measure the degree of classifier's overfitting.

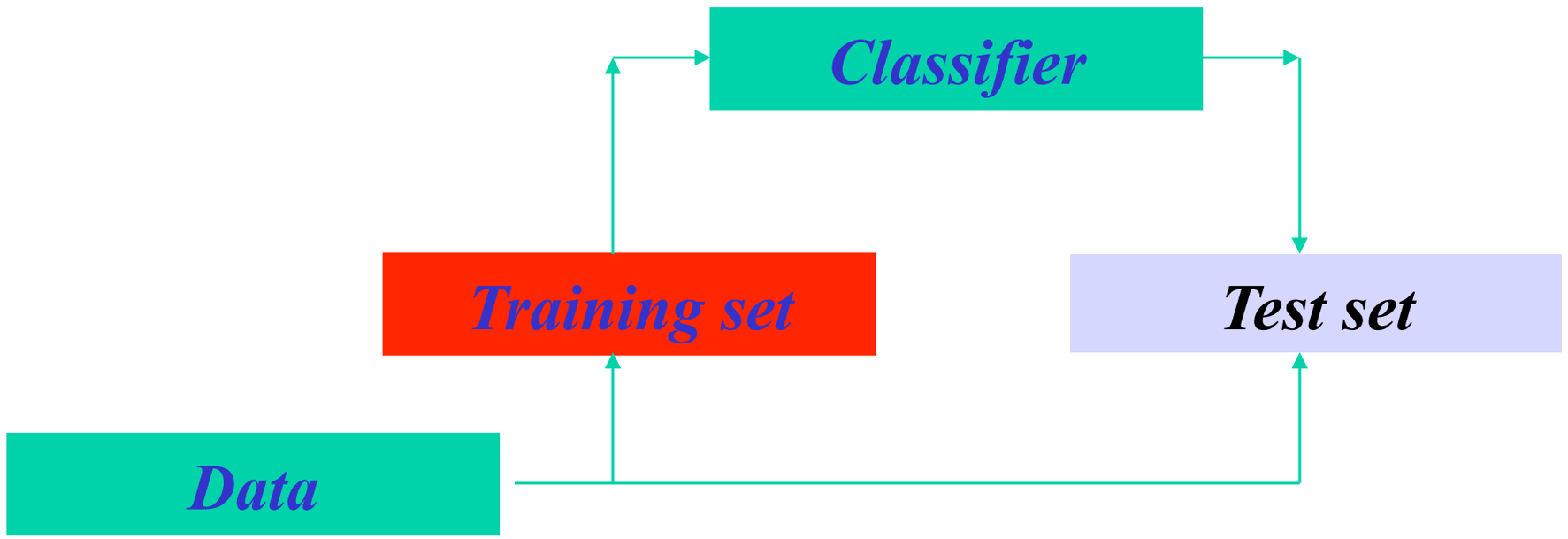
Estimation with TEST Data

- Estimation with independent test data is used when we have plenty of data and there is a natural way to forming training and test data.

- *For example: Quinlan in 1987 reported experiments in a medical domain for which the classifiers were trained on data from 1985 and tested on data from 1986.*

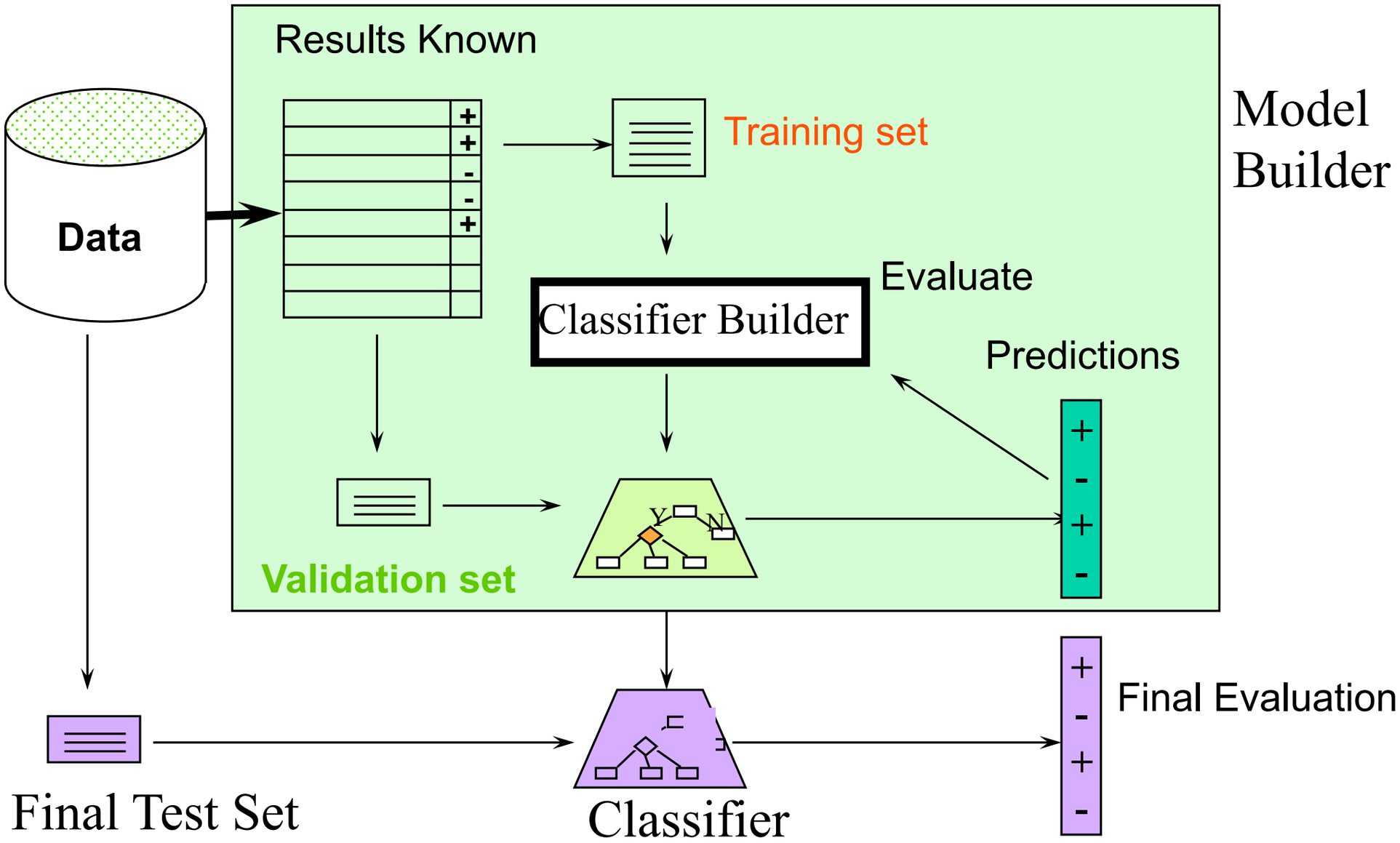
Estimation with Hold Out

- The hold-out method splits the data into training data and test data (usually 2/3 for train, 1/3 for test). Then we build a classifier using the train data and test it using the test data.



- The hold-out method is usually used when we have thousands of instances, including several hundred instances from each class.

Training / Validation / Test



The test data can't be used for parameter tuning!

Well Exploiting your data

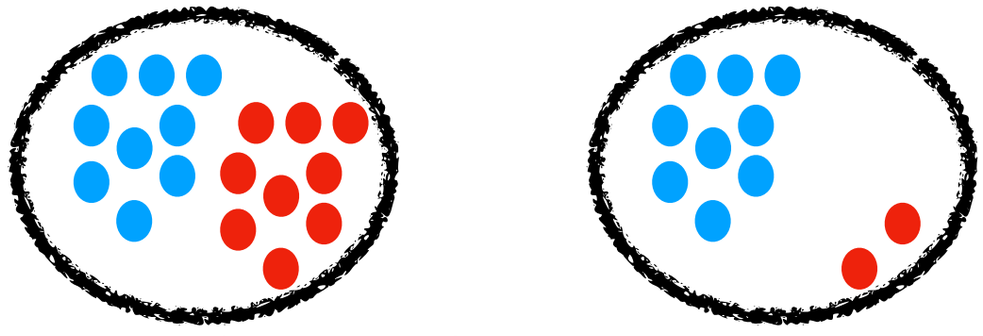
Once evaluation is complete, *all the data* can be used to build the final classifier.

Generally, the larger the training data the better the classifier (but returns diminish).

The larger the test data the more accurate the error estimate.

Stratification to deal with unbalanced data

Unbalanced data



Balanced dataset (w.r.t. classes)

Unbalanced dataset

- The *holdout* method reserves a certain amount for testing and uses the remainder for training.
  - *Usually: one third for testing, the rest for training.*
- For "unbalanced" datasets, samples might not be representative.
  - *Few or none instances of some classes.*
- **Stratified sample: advanced version of balancing the data.**
  - *Make sure that each class is represented with approximately equal proportions in both subsets.*

Adding Reliability to the evaluation: Repeated Hold-Out

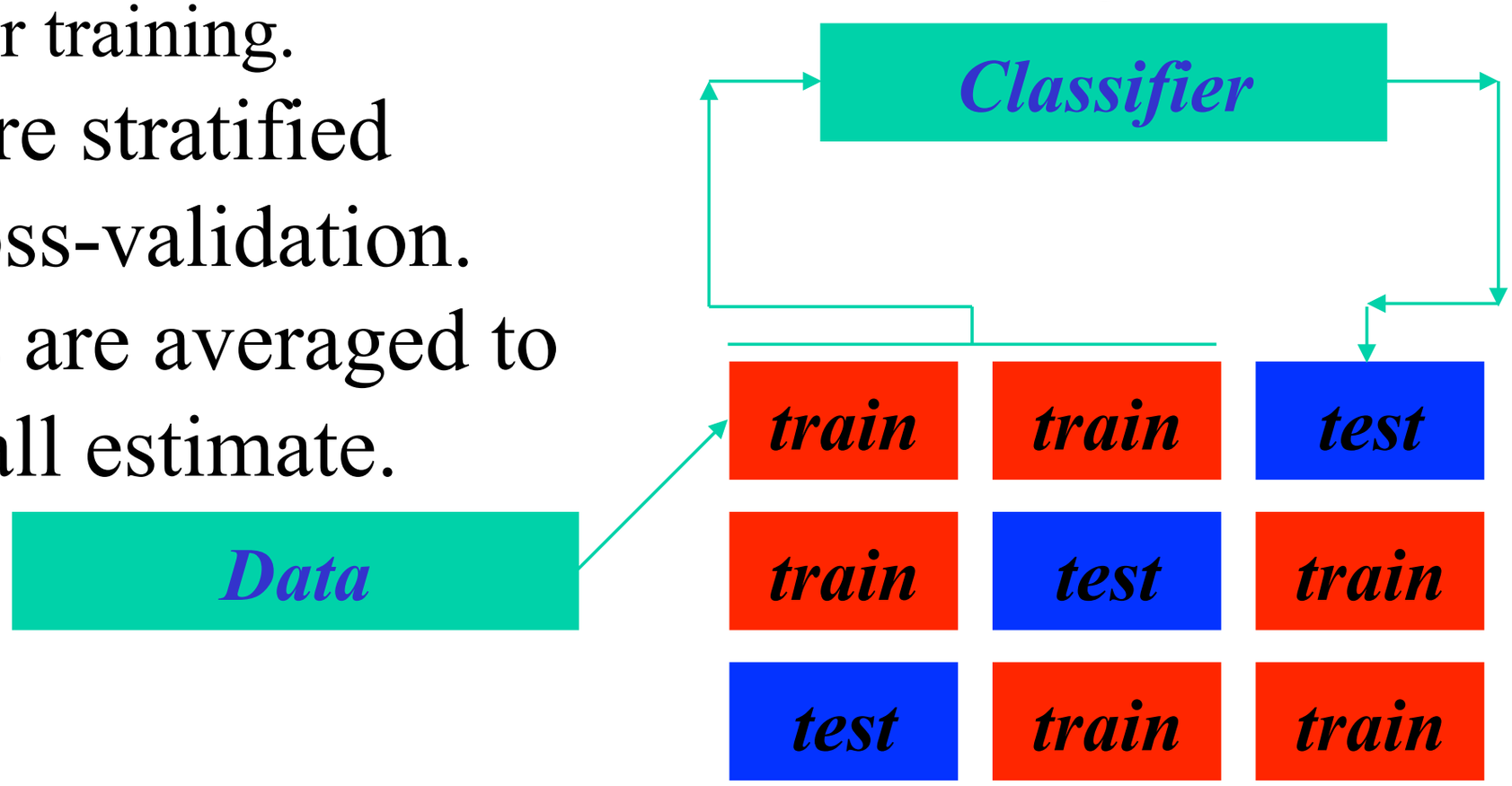
- Holdout estimate can be made more reliable by repeating the process with different subsamples.
  - In each iteration, a certain proportion is randomly selected for training (possibly with stratification).
  - The error rates on the different iterations are averaged to yield an overall error rate.
- This is called the *repeated holdout* method.

Adding Reliability to the evaluation: Repeated Hold-Out

- Still not optimum: the different test sets overlap, but we would like all our instances from the data to be tested at least ones.
- Can we prevent overlapping?

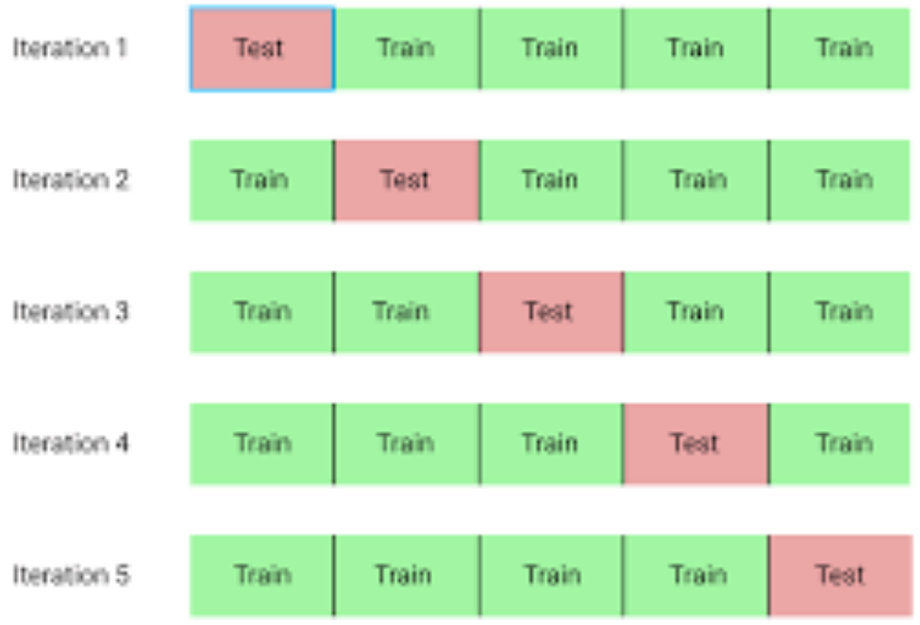
K-Fold Cross Validation

- *k-fold cross-validation* avoids overlapping test sets:
  - *First step*: data is split into *k* subsets of equal size;
  - *Second step*: each subset in turn is used for testing and the remainder for training.
- The subsets are stratified before the cross-validation.
- The estimates are averaged to yield an overall estimate.



K-Fold Cross Validation

Example of
5-Fold Cross Validation



Standard method for evaluation: stratified 10-fold cross-validation.

Why 10? Extensive experiments have shown that this is the best choice to get an accurate estimate.

Stratification reduces the estimate's variance.

Even better: repeated stratified cross-validation:
E.g. ten-fold cross-validation is repeated ten times and results are averaged (reduces the variance).

Leave One Out (Extreme case of KFCV)

- Leave-One-Out is a particular form of cross-validation:
  - Set number of folds to number of training instances;
  - I.e., for *n* training instances, build classifier *n* times.
- Makes best use of the data.
- Involves no random sub-sampling.
- Very computationally expensive.

Leave One Out (Extreme case of KFCV)

- A disadvantage of Leave-One-Out-CV is that stratification is not possible:
  - It *guarantees* a non-stratified sample because there is only one instance in the test set!
- Extreme example - random dataset split equally into two classes:
  - Best inducer predicts majority class;
  - 50% accuracy on fresh data;
  - Leave-One-Out-CV estimate is 100% error!