HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

IT4142E
Introduction to Data Science

Chapter 6: Introduction to Machine Learning

Lecturer:

Muriel VISANI: murielv@soict.hust.edu.vn

Acknowledgements:

Khoat Than
Viet-Trung Tran

Department of Information Systems
School of Information and Communication Technology - HUST

Contents of the course

- Chapter 1: Overview
- Chapter 2: Data scraping
- Chapter 3: Data cleaning, pre-processing and integration
- Chapter 4: Introduction to Exploratory Data Analysis
- Chapter 5: Introduction to Data visualization
- Chapter 6: Introduction to Machine Learning
  - Performance evaluation
- Chapter 7: Introduction to Big Data Analysis
- Chapter 8: Applications to Image and Video Analysis

Goals of this chapter

Goal	Description of the goal
M1	**Understand and be able to design and manage the systems which are based on Data Science (DS)**
M1.2	Identify, compare, and categorize the data type and systems in practice
M1.3	Be able to design systems based on DS in their future organizations

Contents of this chapter

- **Chapter 6: Introduction to Machine Learning**
  - Introduction and definitions
    - What is Machine Learning?
    - Supervised *vs.* unsupervised learning
    - Focus of this chapter
  - Part 1: Unsupervised **clustering**
    - Objective
    - Main issues and useful definitions
    - Methods
      - Partitioning methods
      - Hierarchical methods
      - Grid-based methods
      - Density-based methods
    - Performance evaluation
  - Part 2: Supervised **classification**
    - Objective
    - Methods
    - Performance evaluation
  - Summary
  - Homework

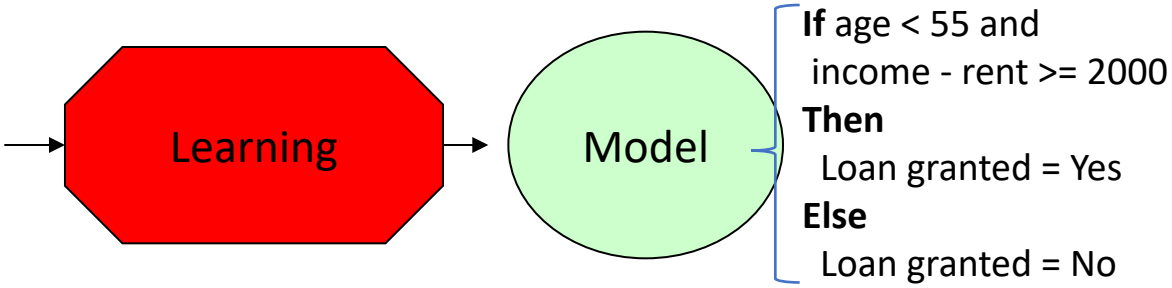
Recall: supervised learning

- **Supervised** learning methods
  - The learning dataset contains the values of the response variable(s)
  - Example

Step 1: Learning the model (*a.k.a.* learning step or training step)

Learning dataset

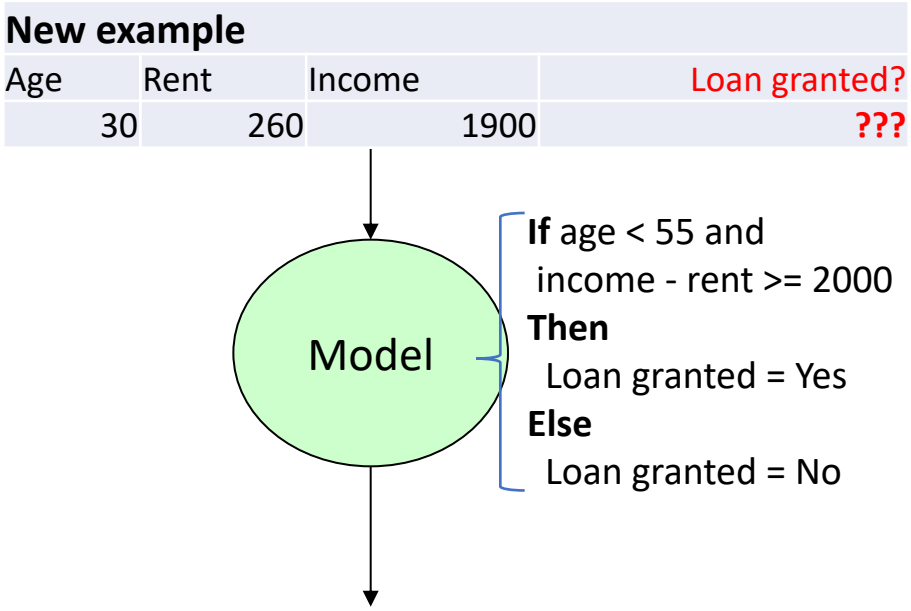
Age	Rent	Income	Loan granted?
36	0	1299	Yes
55	240	2500	No
40	768	3000	Yes
39	0	2000	Yes
44	334	512	No
26	631	722	No



Recall: supervised learning

- **Supervised** learning methods
  - The learning dataset contains the values of the response variable(s)
  - Example

Step 2: Applying the model (prediction)



The model cannot always be expressed using rules
- it depends on the model
- *e.g.* decision trees give rules but neural networks don't

Supervised classification

- Let $\boldsymbol{y}=(y^1,\ldots, y^p)^T$ be the vector of explanatory variables of an observation
- Let *c* be the class of the observation ***y*** (response variable)
- For each object in the Learning Dataset, we know the joint values of the $(\boldsymbol{y}; c)$ (by definition of supervised classification)
- For any new sample (to classify), we only know ***y***, and we want to infer its class *c*
- We're aiming at learning a **classifier** *f* that allows:
  - to classify the training dataset as correctly as possible (fitting)
  - to best predict the class of new samples (generalization)

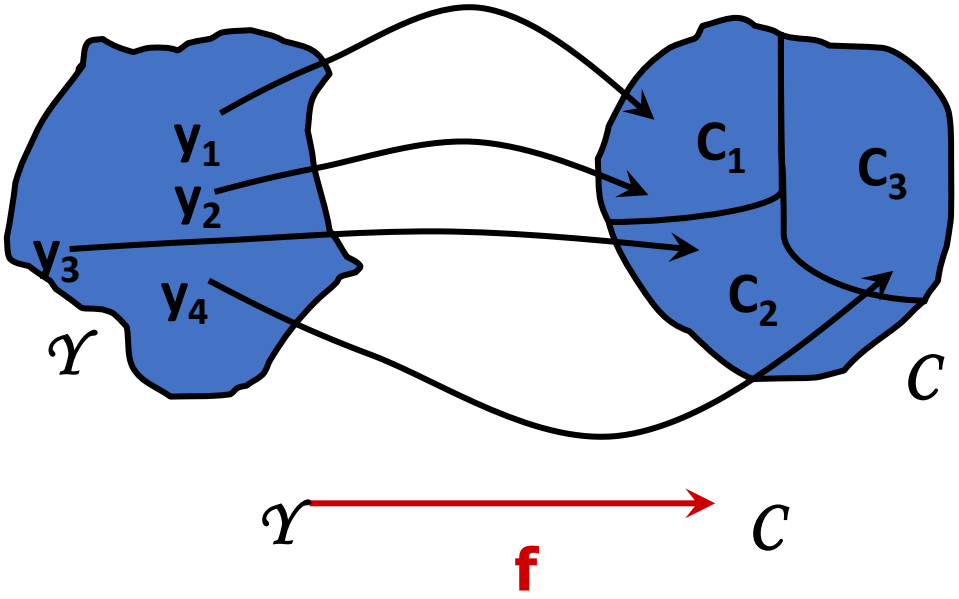
Supervised classification

- Therefore, we are looking for a function

$$f: \mathcal{Y} \to \mathcal{C}$$
$$Y \mapsto f(Y)$$

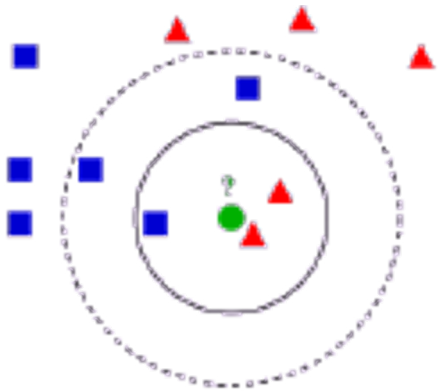
  - Such as *f (**y**)* = *c* with $\mathcal{C} = \{c_1, \ldots, c_k\}$ being the set of all possible classes (**modalities** of the response variable)

  - *f* is often (wrongfully) called **classifier**



Supervised classification: k-nn

- **Example** of a basic classifier: *k* nearest neighbours (*k*-nn)
  - The training dataset is labeled with the corresponding class
    - Blue square or red triangle
  - The new sample in green, $\boldsymbol{y}=(y^1,\ldots, y^p)^T$, with unknown class, is assigned to the majority class among its *k* nearest neighbours from the training dataset



k: classifier parameter
If k = 3, the new sample is labelled « triangle »
If k = 5, the new sample is labelled « square »

Supervised classification: k-nn

☺ Extremely simple classifier: does not even rely on learning a model

☺ Simple to implement

☹ Can be applied only on **numeric** explanatory variables (not to categorical variables)

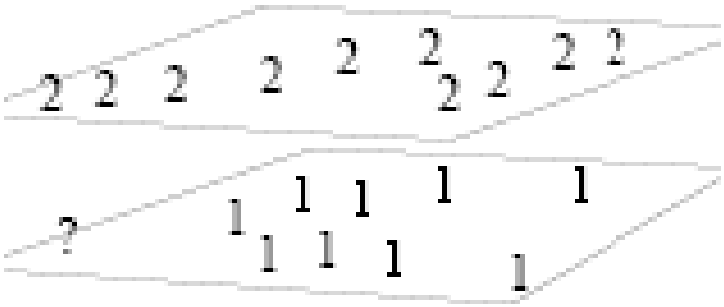
☹ Sensitive to outliers

☹ **Very** dependent on its parameter *k*

☹ **Very** dependent on the distance used

Quizz

- Let's consider the following example, in 3D, with 2 classes (1 & 2)
- The training data is labeled with its class (1 or 2)
- The new sample is marked as "?"
- Let us consider the 2 nearest neighbour classifier
- **Q1:** what is the predicted class for "?", if we use an Euclidean distance?
  - Response:
- **Q2:** what is the predicted class for "?", if we use the Mahalanobis distance?
  - Response:
- **Q3:** which distance should we choose, given the training data distribution?
  - Response:



Supervised classification – main issues

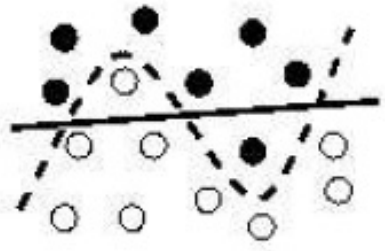
- In a real-world application, a classifier won't be able to properly classify all the observations
  - Real-life is not perfect!
- Hence the search for a **compromise** between **fitting** and **generalization**
- It is **easy** to assess the **fitting** performances of the classifier
  - by comparing the real class *c* with the class predicted *f(**y**)*...
  - ... for each example of the training dataset
- It is **more difficult** to assess the **generalization** capacity of the classifier
  - Indeed, on new samples, we know the explanatory variables ***y*** and we can infer the predicted class *f(**y**)…*
  - but we don't know the real class *c*!
- We'll see later on in this chapter how we can estimate generalization capability of the model
  - But, this point leads us to the issue of **over-fitting**

Supervised classification – over-fitting

- The issue of over-fitting occurs often when we don't have enough observations in the training dataset
  - If we try too hard to perfectly fit the classifier to the training data…
  - … then the classifier will be perfectly fitting the training data…
  - … but might not be good for new samples with the same distribution!!!
- The model has **learnt « by heart »** the learning dataset and is incapable of using its knowledge in a slightly different context
  - The learning dataset is just a sub-sample of the more general reality

Supervised classification – over-fitting

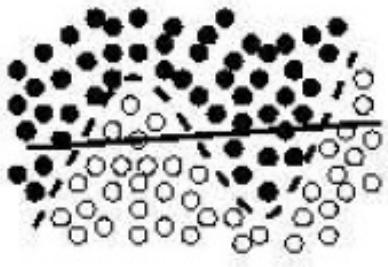
- Illustrative example:



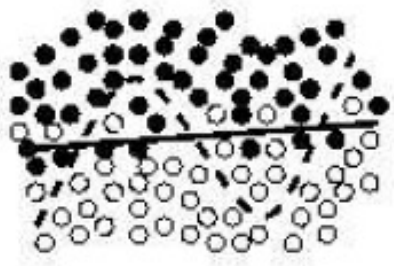
Training dataset:
Few observations
2 classes (white / black)

Problem: which is the best classifier (here frontier between classes)?
The dashed curve or the solid line?

- Well, it all depends on the underlying distribution of the data!



In this case, the dashed curve is better



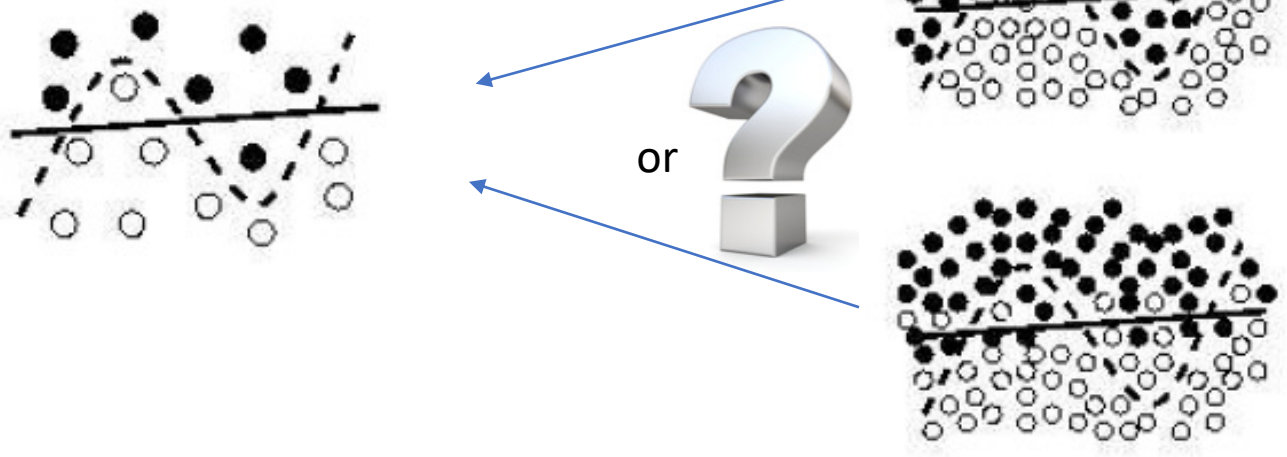
In this case, the solid line is better

Supervised classification – over-fitting

- Illustrative example:

Possible underlying distributions

Training dataset:
Few observations

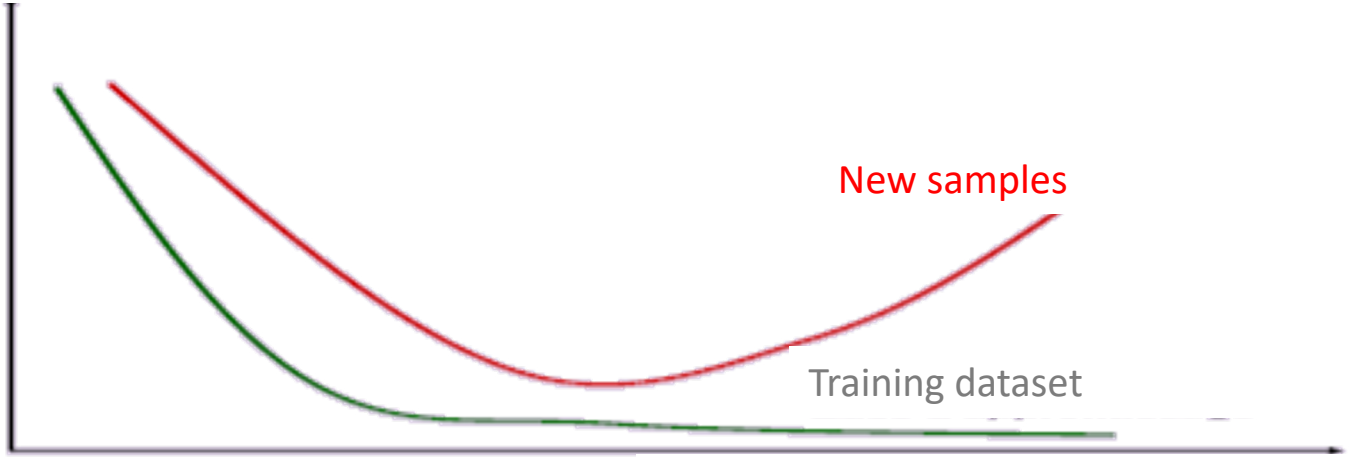


- Because we generally don't know the underlying distribution, it is sometimes better to choose a model that is simpler and does not fit perfectly the training data (here the solid line)
  - To avoid over-fitting (learning by heart) the training data

Supervised classification – over-fitting

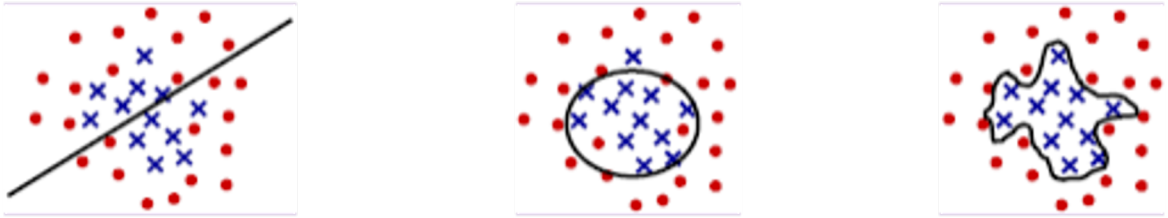
- Another illustrative example:
  - Which classifier is the best?

Classification error rate



Increasing complexity of the classifier

Training dataset with different classifiers



Supervised classification – over-fitting

- If we evaluate the performance of the classifier on the training dataset
  - Called apparent (or resubstitution) error rate
  - Not representative of the classifier's generalization capacity
    - Too optimistic
  - Therefore, we'll have to use a separate dataset for performance evaluation usually called **test dataset**)
    - *E.g.* a random subsample of the training dataset, that we **do not** use for learning the classifier
      - We only use it for assessing its generalization performances
- More details about this witll be given later in in this chapter (section « performance evaluation »)
  - First of all, let's present some of the **many many** methods that can be used for supervised classification

Supervised classification

Methods

Supervised classification methods

- There are three main types of supervised classification methods:
  1. Probabilistic methods
     - Aim: to minimize the classification error rate based on probability estimates
     - For categorical or numeric attributes (assumptions on their distribution)
     - Examples: Maximum likelihood rule, Bayesian classifiers...
  2. Symbolic methods
     - Aim: to infer the most likely possible decision-making rules
     - Often, for categorical or « sliced » (discretized) numeric attributes
     - Examples: decision trees, rule extraction...
  3. Statistical methods
     - Aim: to minimize an estimate of the classifier's generalization error rate
     - Often for quantitative attributes (assumptions about their distribution)
     - Examples: discriminant analysis, SVM, (deep) neural networks...

Supervised classification

Probabilistic methods

Probabilistic methods

- Notations :
  - ***y***: vector of explanatory variables (description)
  - P[***y***]: probability of observing the description ***y***
  - P[*c*]: probability for an observation to belong to class *c*
  - P[*c*/***y***]: probability, for an observation with description ***y,*** to belong to class *c*:

$$P[c/y] = \frac{P[\ y\ \&\ c\ ]}{P[y]}$$

  - P[***y***/*c*]: probability, for an observation belonging to class *c,* to have description ***y***
  - Problem: we are looking for P[*c*/***y***] (this will define our classifier) but we cannot estimate it easily (unknown underlying distribution of the classes)
    - But, we can estimate P[***y***/*c*] on the training dataset!
  - So, we're going to use Bayes formula:

$$P[c/y] = \frac{P[y/c]P[c]}{P[y]}$$

Bayes classifier

- Bayes classifier:
  - To each observation ***y***, we assign the class *c* such that P[***y***/*c*]P[*c*] is maximum
  - Can be applied to either categorical or numeric variables, by making assumptions
    - *E.g.* the variable is Bernouilli or Gaussian
  - **Theorem**: the Bayes classifier is optimal on the training dataset
    - But, prone to overfitting…

Bayes classifier

- **Exercise**:
  - A (very clever) child wants to predict his Mom's answer when he'll ask her if he can play outside after school
  - He figured this depends a lot on if he finished his homework, or not
  - He collected data for 10 days, and made the assumption that « Homework » is a **Bernouilli variable**

Day	H: did I finish my homework before asking?	'c' : decision
1	1	Yes
2	1	Yes
3	1	Yes
4	1	Yes
5	1	Yes
6	0	Yes
7	0	No
8	0	No
9	1	No
10	0	No

Bayes classifier

- **Exercise**:
  - Give the result of the Bayes classifier f_{Bayes} under the form $f_{Bayes}(H=0)=xx$ and $f_{Bayes}(H=1)=xx$, where xx must be replaced by « yes » or « no »
  - Give the apparent classification error (# of misclassified observations / # of observations) using this rule

Day	H: did I finish my homework before asking?	'c' : decision
1	1	Yes
2	1	Yes
3	1	Yes
4	1	Yes
5	1	Yes
6	0	Yes
7	0	No
8	0	No
9	1	No
10	0	No

Bayes classifier

- **Solution**:

Day	H: did I finish my homework before asking?	'c' : decision
1	1	Yes
2	1	Yes
3	1	Yes
4	1	Yes
5	1	Yes
6	0	Yes
7	0	No
8	0	No
9	1	No
10	0	No

Bayes classifier

- **Multi-variable Bayes classifier**
  - The child figures it does not only depends on his homework. So now, he wants to take into account other variables:
    - His Mom's mood (M), the weather (W), if he had a snack (S)

Day	H: did I finish my homework before asking?	M: is Mom in a good mood?	W: is it sunny (1) or rainy (0)?	S: Did I have a snack?	'c' : decision
1	1	1	1	1	Yes
2	1	0	1	1	Yes
3	1	0	1	0	Yes
4	1	0	1	0	Yes
5	1	1	1	0	Yes
6	0	1	1	1	Yes
7	0	0	0	0	No
8	0	1	1	0	No
9	1	1	0	1	No
10	0	0	1	1	No

Bayes classifier

- **Question:**
  - How many configurations are possible?
    - Response: 16
  - Did we observe all of them in the learning set?
    - Response: No
  - What can we do then?
    - Is putting their estimated probabilities at 0 fair?
      - Response: No, because the corresponding configuration could never be predicted, even though it is possible
    - **Solution: naive Bayes**

Naive Bayes classifier

- Naive Bayes Classifier
  - Assumption of **independence** of explanatory variables y_i **conditionally** to their class of belonging c:
  - For each y, we assign the class c s.t. P[y/c]P[c] is maximum, where

$$P[Y = y/c] = P[Y^1 = y^1/c]P[Y^2 = y^2/c] \ldots P[Y^p = y^p/c]$$

  - Simple to implement, relatively good performance
  - Can solve non-linearly separable (so relatively complex) classification problems
  - Based on a generally **false** hypothesis
    - But often gives good results in practice
    - Serves as a benchmark for evaluating more elaborate methods

Homework

- By hand, calculate / give the Bayes rule for the couple of explanatory variables H and W

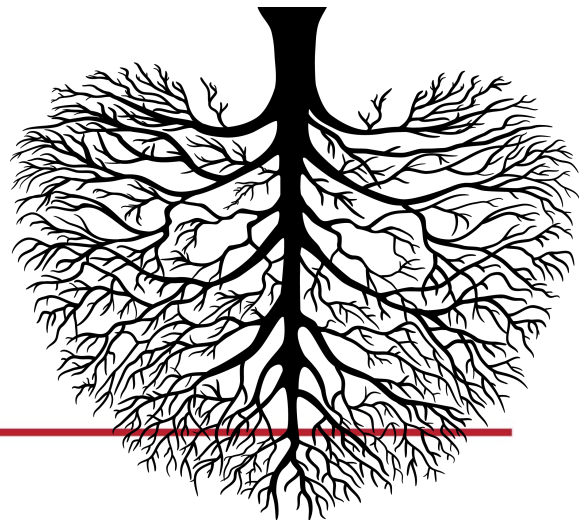
Day	H: did I finish my homework before asking?	M: is Mom in a good mood?	W: is it sunny (1) or rainy (0)?	S: Did I have a snack?	'c' : decision
1	1	1	1	1	Yes
2	1	0	1	1	Yes
3	1	0	1	0	Yes
4	1	0	1	0	Yes
5	1	1	1	0	Yes
6	0	1	1	1	Yes
7	0	0	0	0	No
8	0	1	1	0	No
9	1	1	0	1	No
10	0	0	1	1	No

Supervised classification

Symbolic methods

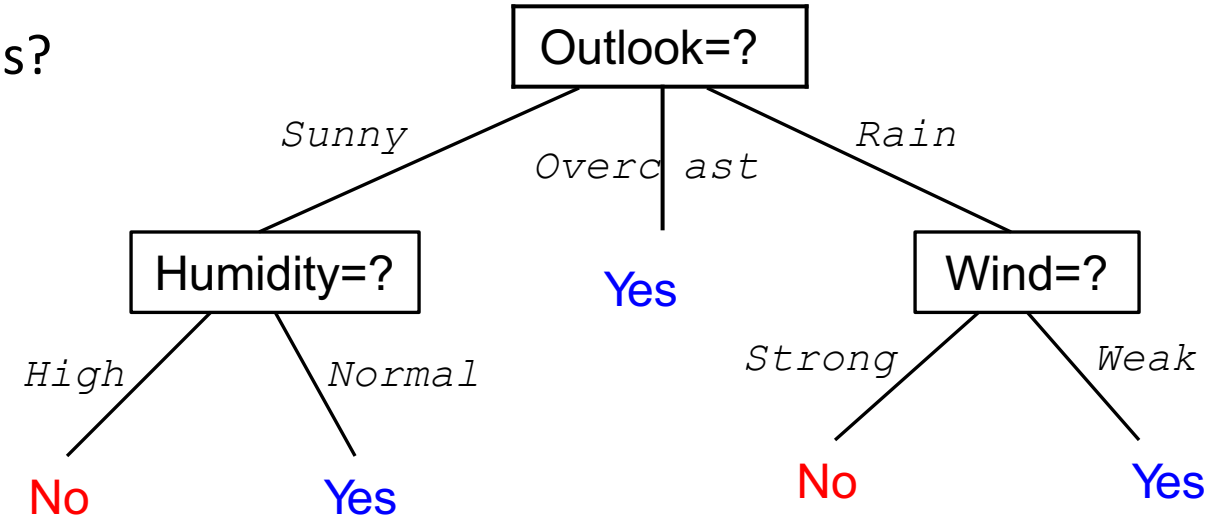
1. Decision tree

- **Symbolic methods**
  - Aim: to infer the most likely possible decision-making rules
  - Often, for categorical or discretized numeric attributes
- Example of symbolic method: Decision tree
  - To represent a function by using a tree.
- Each decision tree can be interpreted as a set of rules of the form: IF-THEN
- Decision trees have been used in many practical applications.



Examples of a decision tree

- Playing tennis?



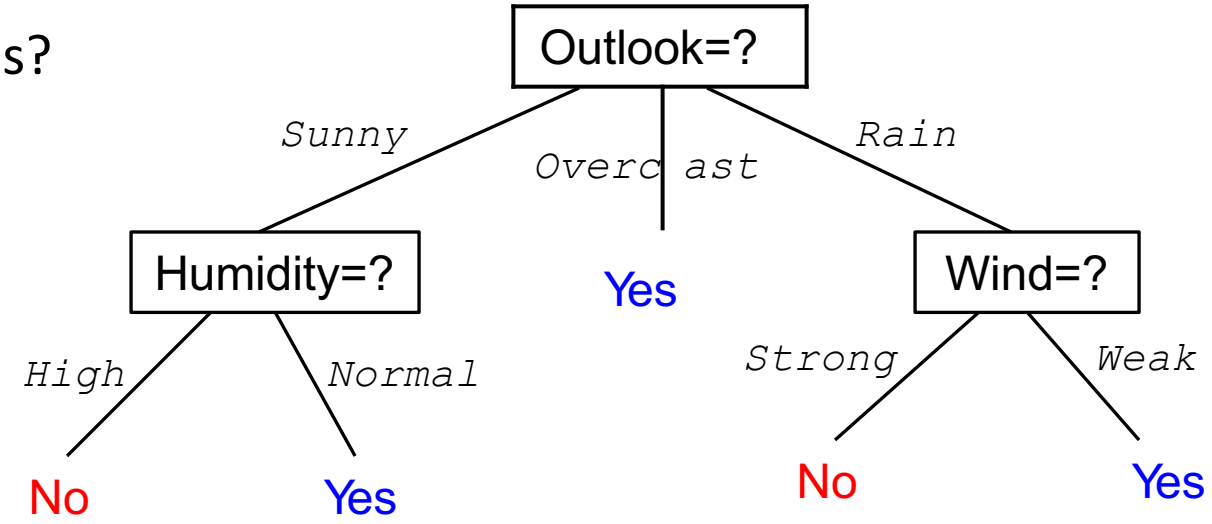
- (Outlook=`Overcast`) → Yes
- (Outlook=`Rain`, Wind=`Strong`)
  → No
- (Outlook=`Sunny`, Humidity=`High`)
  → No

Tree representation

- *Each* **internal node** represents an attribute for **testing** the incoming data
- Each **branch/subtree** *of a node* corresponds to a **value** of the attribute of that node (**modality for categorical variables**)
- Each **leaf** node represents a **class** label.
- Once a tree has been learned, *we can predict the label* for a new instance by using its explanatory attributes to travel from the root downto a leaf.
  - The label of the leaf will be used to assign to the new instance.

Examples of a decision tree

- Playing tennis?

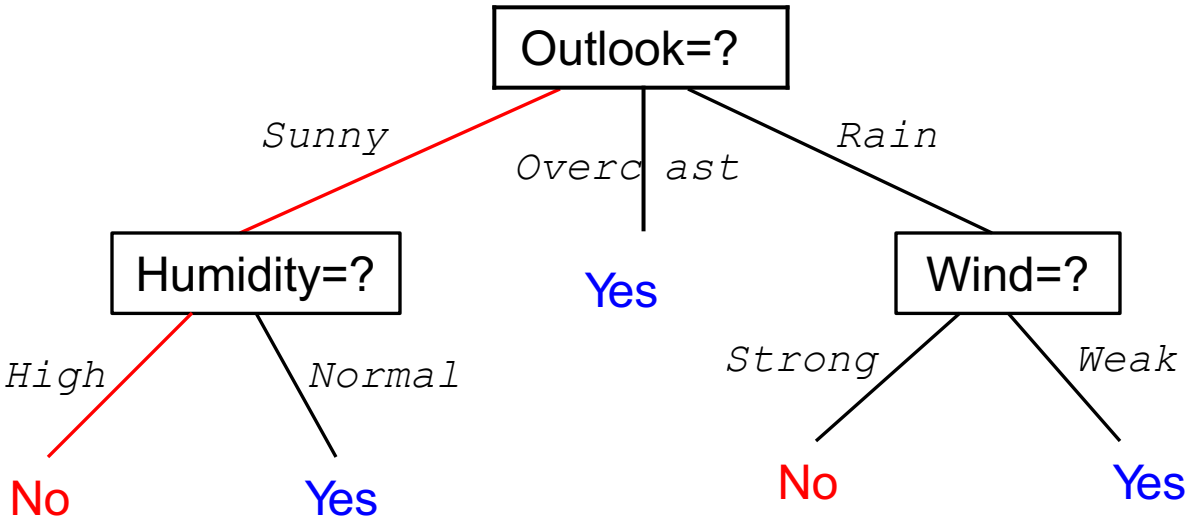


- Quizz:
  - Today the wind is weak but the humidity is high, as it's raining.
  - Will my teammates want to play tennis?
    - Response: Yes

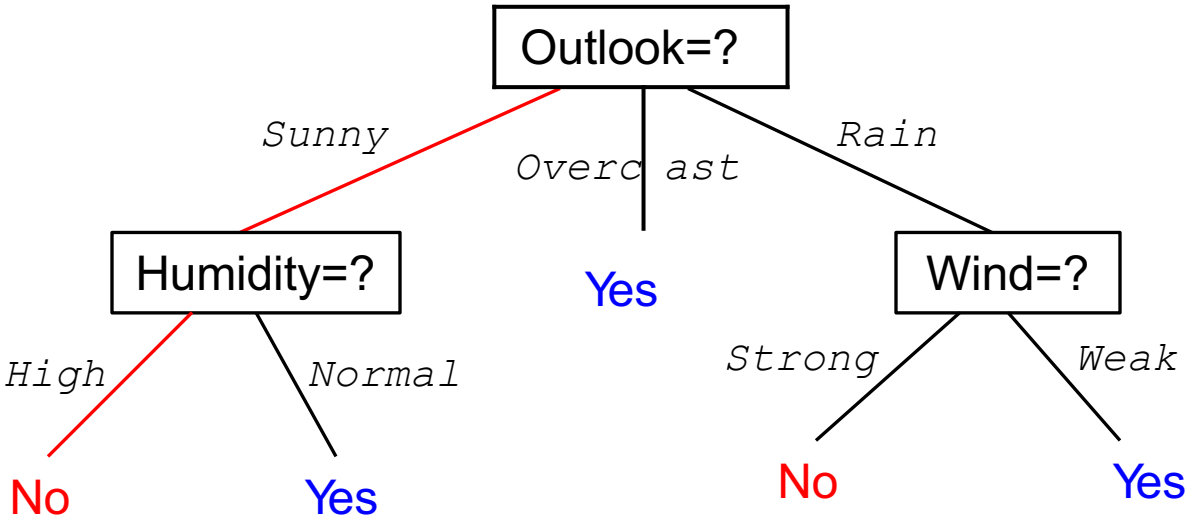
Tree representation

- Each path from the root to a leaf is a *conjunction/AND* of the tested attributes

- A decision tree itself is a *disjunction/OR* of those conjunctions



Representation by a disjunction



[("Outlook" is Sunny) ∧ ("humidity" is High)] ∨

[("Outlook" is Rain) ∧ ("Wind" is Strong)] ∨

[("Outlook" is Sunny) ∧ ("humidity" is Normal)] ∨

[("Outlook" is Overcast) ∨

[("Outlook" is Rain) ∧ ("Wind" is Weak)]

2. Learning a decision tree by ID3

- ID3 (Iterative Dichotomiser 3) is a greedy **algorithm** which was proposed by Ross Quinlan in 1986
- It uses a top-down approach.
- At each node N, select a test attribute A which can help us best do classification for the data in node N
  - *Generate a branch for each value of A, and then separate the data into its branches accordingly*
- Grow the tree until:
  - *It classifies correctly all the training data; or*
  - *All the attributes are used.*

2. Learning a decision tree by ID3

- Notes: different from other algorithms (CART for instance)
  - *Each node can have multiple children*
    - *E.g. all possible modalities of a categorical variable…*
    - *… or even a combination of multiple variables*
  - *each attribute can only appear at most once in any path of the tree*

The ID3 learning algorithm

ID3_alg(Training_Set, Class_Labels, Expl_Attributes)

Generate the Root of the tree

If all of *Training_Set* belong to class c, then Return Root as leaf with label c

If Expl_*Attributes* is empty, then
    Return Root as leaf with label c = **Majority_Class_Label**(*Training_Set*)

A ← a set of *Expl_Attributes* that are best discriminative for *Training_Set*

Let A be the test attributes of Root

For each possible value v of A

    Generate a branch of Root which corresponds with v.

    Determine $Training\_Set_v = \{ x \text{ in } Training\_Set \mid x_A = v \}$

    If ($Training\_Set_v$ is empty) Then

        Generate a leaf with class label c = **Majority_Class_Label**(*Training_Set*)

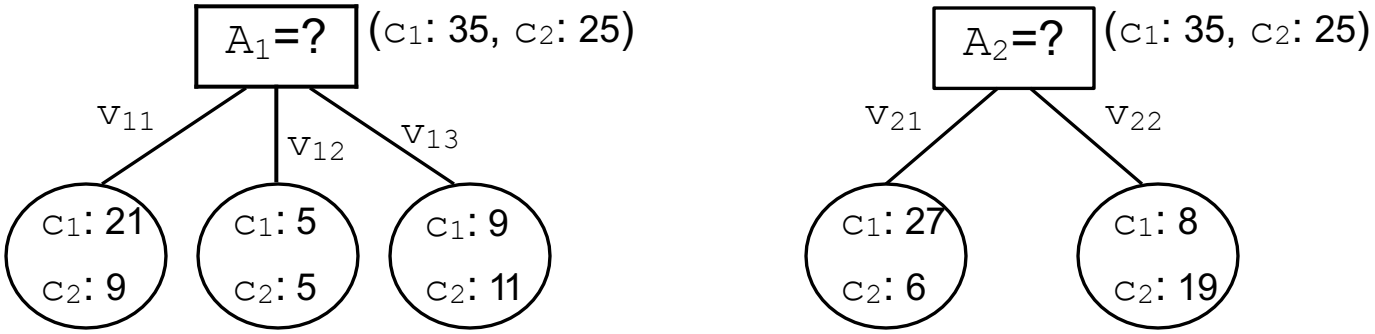
    Else

        Generate a subtree by **ID3_alg**($Training\_Set_v$, Class_Labels, Expl_Attributes \{A})

Return Root

How to choose the test attributes?

- At each node, how can we choose a set of test attributes?
  - These attributes should be *discriminative*, i.e., can help us classify well the data inside that node.
- How to know an attribute to be discriminative?
- Ex: assuming 2 classes in the data, which of A_1 and A_2 should be selected as the test attribute?



- ***Information gain*** can help, using Gini index (CART algorithm) or entropy (ID3 algorithm)

Information gain: entropy

- Entropy measures the impurity/inhomogeneity of a set.
- Entropy of a set S with c classes can be defined as:

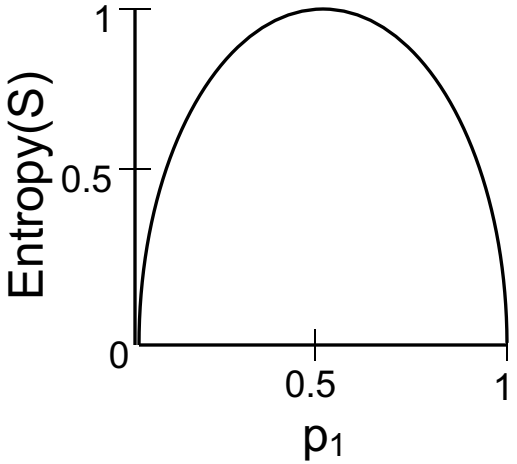
$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

  - Where p_i is the proportion of instances with class label i in S; and $0.\log_2 0 = 0$ as a convention; $p_1+p_2+\ldots+p_c = 1$
- For 2 classes: entropy(S) = $- p_1\log_2 p_1 - p_2\log_2 p_2$
- **Idea**: look for the attribute that makes the class distribution most « peaky » amongst its children

Information gain: entropy example

- S consists of 14 examples for which 9 belong to class c_1 and 5 belong to class c_2.
- So the entropy of S is:

Entropy(S)
$= -(9/14).\log_2(9/14) -(5/14).\log_2(5/14)$
≈ 0.94



- Entropy = 0 if all examples in S have the same label.
- Entropy = 1 if the two classes in S are equal in size.
- Otherwise, entropy will always belong to (0, 1).

Information gain

- *Information gain* of an attribute in S:
  - Measures the reduction of entropy if we divide S into subsets according to that attribute.
- Information gain of attribute A in S is defined as:

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

  - Where Values(A) is the set of all values of A, and $S_v = \{\mathbf{x} \mid \mathbf{x} \text{ in } S, \text{ and } x_a = v\}$
- The **second term** in Gain(S,A) measures the *information (diversity) remaining* when S is divided into subsets according to the values of A.
- Meaning of Gain(S,A): *the average amount of « purity » that we gain when dividing S according to A.*

Information gain: example (1)

- A set S of observations about a team playing tennis.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

[Mitchell, 1997]

Information gain: example (2)

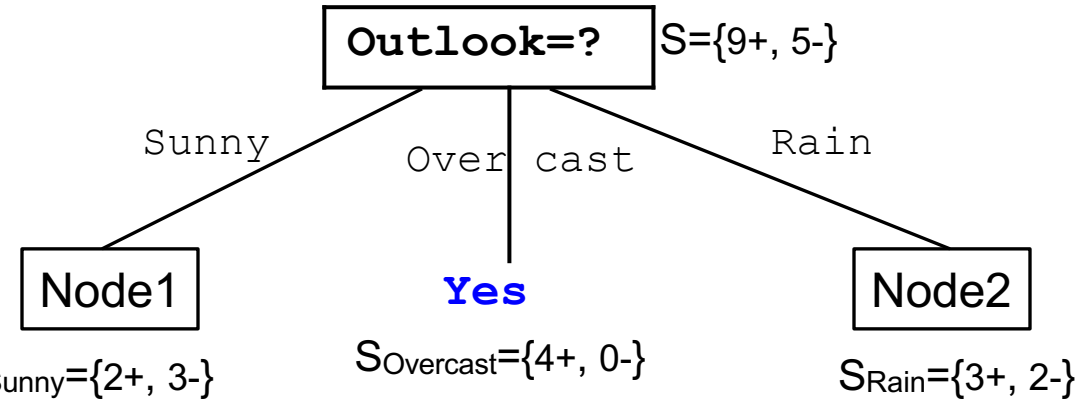
- What is Gain(S, Wind)?
- Wind has two values: Strong & Weak
- S = {9 examples with label Yes, 5 examples with label No}
- S_{Weak} = {6 examples with label Yes and 2 examples with label No, having Wind=Weak}
- S_{Strong} = {3 examples with label Yes, 3 examples with label No, having Wind=Strong}
- So: $Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$

$$= Entropy(S) - \frac{8}{14} Entropy(S_{Weak}) - \frac{6}{14} Entropy(S_{Strong})$$

$$= 0.94 - \frac{8}{14} * 0.81 - \frac{6}{14} * 1 = 0.048$$

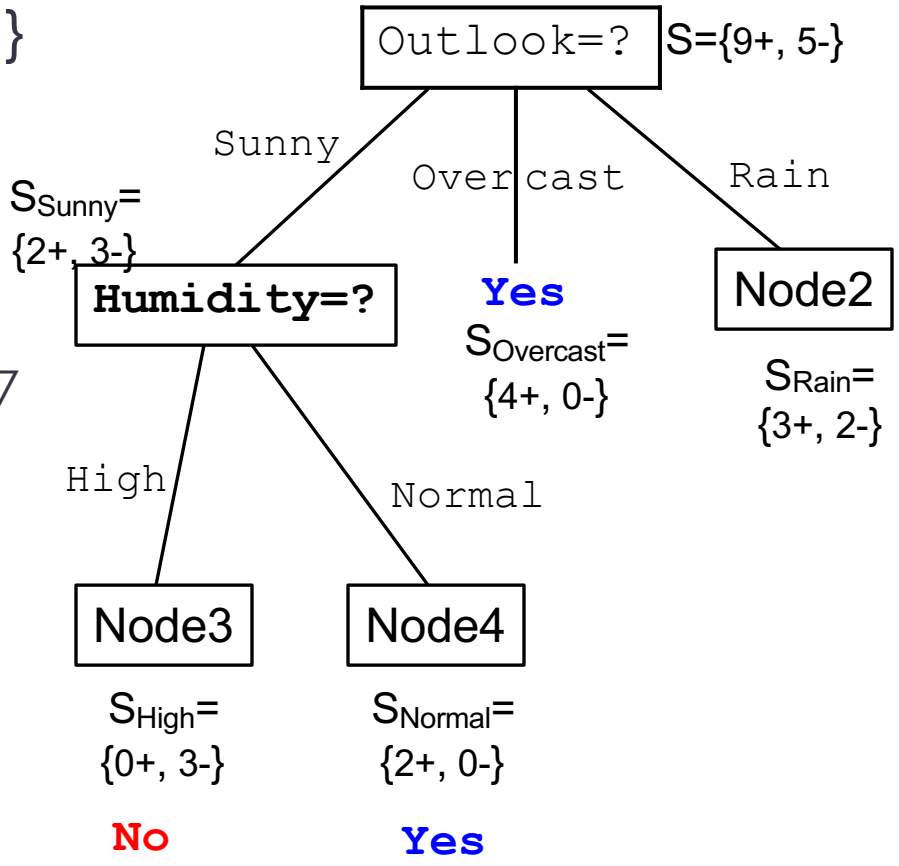
ID3: example (1)

- At the root, which one of {Outlook, Temperature, Humidity, Wind} should be the test attribute?
  - Gain(S, **Outlook**) = ... = **0.246**
  - Gain(S, Temperature) = ... = 0.029
  - Gain(S, Humidity) = ... = 0.151
  - Gain(S, Wind) = ... = 0.048
- So, Outlook is selected as the test attribute at the root level.



ID3: example (2)

- At Node 1, which of {Temperature, Humidity, Wind} should be the test attribute?
  - Note: Outlook is left out
  - Gain(S_{Sunny}, Wind) = ... = 0.019
  - Gain(S_{Sunny}, Temperature) =...= 0.57
  - Gain(S_{Sunny}, **Humidity**) = ... = **0.97**
- So, Humidity is selected to divide Node 1.
- Node 3 & Node 4 are labelled with their classes (because they are « pure »)
- And So on for Node 2…



Some issues of ID3

a) The learnt trees may overfit the training data.

b) How to work with real (numeric, continuous) attributes?

- Many applications have real inputs.

c) How to deal with missing values?

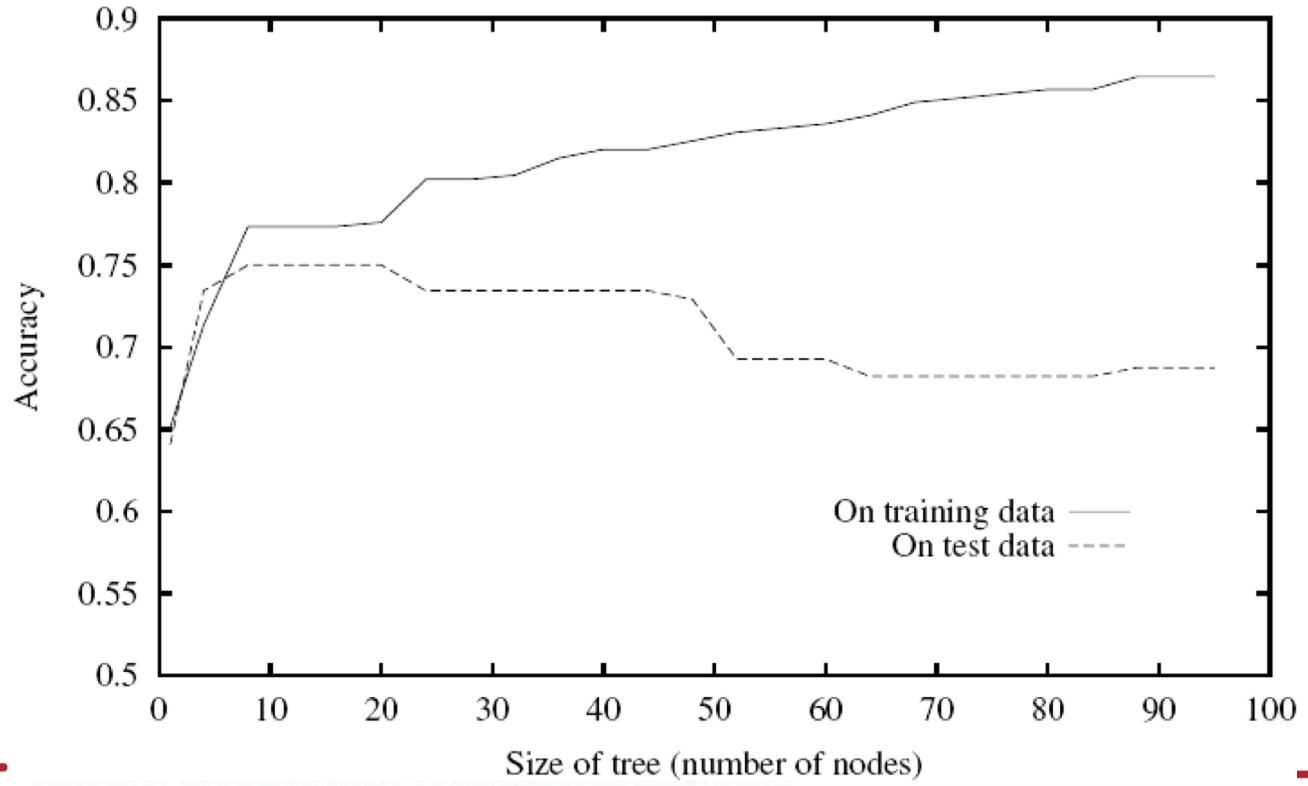
- Missing-value is an inherent problem in many practical applications.

d) Is there any better measure than information gain?

e) How to enclose the cost of attributes in ID3?

(a) Overfitting in ID3

- An example: continuing to grow the tree can improve the accuracy on the training data, but perform badly on the test data (same idea as on slide 12 of this course)



[Mitchell, 1997]

(a) Overfitting in trees: solutions

- 2 solutions:
  - *Stop learning early:* prevent the tree before it fits the training data perfectly.
  - *Prune the full tree:* grow the tree to its full size, and then post-prune the tree.
- It is hard to decide when to stop learning
  - defining a **stopping criterion**: max depth, min # samples /node, min « purity » per node…
- Post-pruning the tree empirically results in better performance. But
  - How to decide the good size of a tree?
  - When to stop pruning?
  - We can use a **validation set** to do pruning

(b-c) ID3: missing or real values

b) How to work with real attributes?

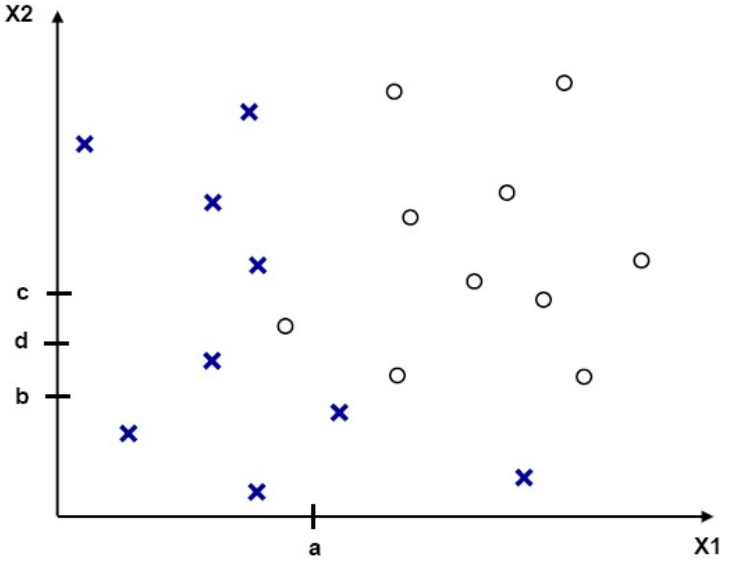
- Real attributes/features are popular in practice.
- One way is to *discretization*, i.e., transforming a real attribute into a discrete one by dividing the domain of that attribute into a set of intervals.
  Ex: [0, 1] → {[0, 0.25); [0.25, 0.5); [0.5, 0.75); [0.75, 1]}

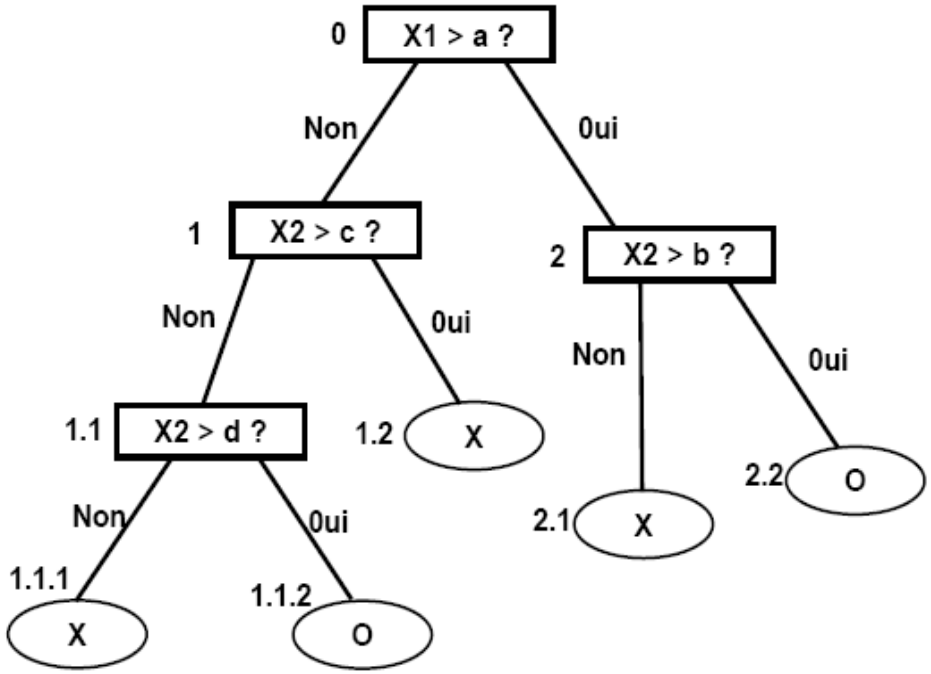
c) How to deal with missing values?

- Missing values are inherent in practical applications.
- An observation **x** may not have a value x_A.
- *Solution 1:* fill in x_A as the modal value of A in the training data.
- *Solution 2:* fill in x_A as the modal value of A in the training data which belong to the same class with **x**.

Exercise

- Let's consider the following learning dataset and associated decision tree
- Draw (in the left window) the tree frontiers





Homework

- Based on the Bayes' child example, build the **complete** tree using CART tree algorithm (using Gini instead of entropy, and binary tree with only 1 variable at each node)
- Where would you propose to prune it?

2. Random forests

- Random forests (RF) is a method by Leo Breiman (2001) for both classification and regression.
- **Main idea:** prediction is based on combination of many decision trees, by *taking the modal class (or average, for regression) of all individual predictions.*
- Each tree in RF is simple but based on a random selection of the learning records AND attributes.
  - On average, can diminish the effect of outliers
  - On average, can diminish the effect of redundant / dirty variables

2. Random forests

- RF currently is one of the most popular and accurate methods [Fernández-Delgado et al., 2014]
  - It is also very general.
- RF can be implemented easily and efficiently.
- It can work with problems of very high dimensions, without overfitting ☺
- However, little is known about its theoretical properties ☹
- It is less « interpretable » than a single tree ☹

2. RF: algorithm

- **Input**: training data D
- **Learning**: grow K trees as follows
  - Generate a training set D_i by sampling with replacement from D
  - Learn the i^{th} tree from D_i:
    - At each node:
      - Select randomly a subset S of attributes.
      - Split the node into subtrees according to S.
    - Grow this tree upto its largest size (in general, without pruning).
- **Prediction**: taking the modal class (or average if regression) of all predictions from the individual trees.

2. RF: practical performance

- RF is extensively compared with other methods
  - By Fernández-Delgado et al. (2014).
  - Using 55 different problems.
  - Using average accuracy (μ^P) as a measure.

No.	Classifier	μ^P	No.	Classifier	μ^P
1	rf_t	91.1	11	Bagging_LibSVM_w	89.9
2	parRF_t	91.1	12	RandomCommittee_w	89.9
3	svm_C	90.7	13	Bagging_RandomTree_w	89.8
4	RRF_t	90.6	14	MultiBoostAB_RandomTree_w	89.8
5	RRFglobal_t	90.6	15	MultiBoostAB_LibSVM_w	89.8
6	LibSVM_w	90.6	16	MultiBoostAB_PART_w	89.7
7	RotationForest_w	90.5	17	Bagging_PART_w	89.7
8	C5.0_t	90.5	18	AdaBoostM1_J48_w	89.5
9	rforest_R	90.3	19	Bagging_REPTree_w	89.5
10	treebag_t	90.2	20	MultiBoostAB_J48_w	89.4

Supervised classification

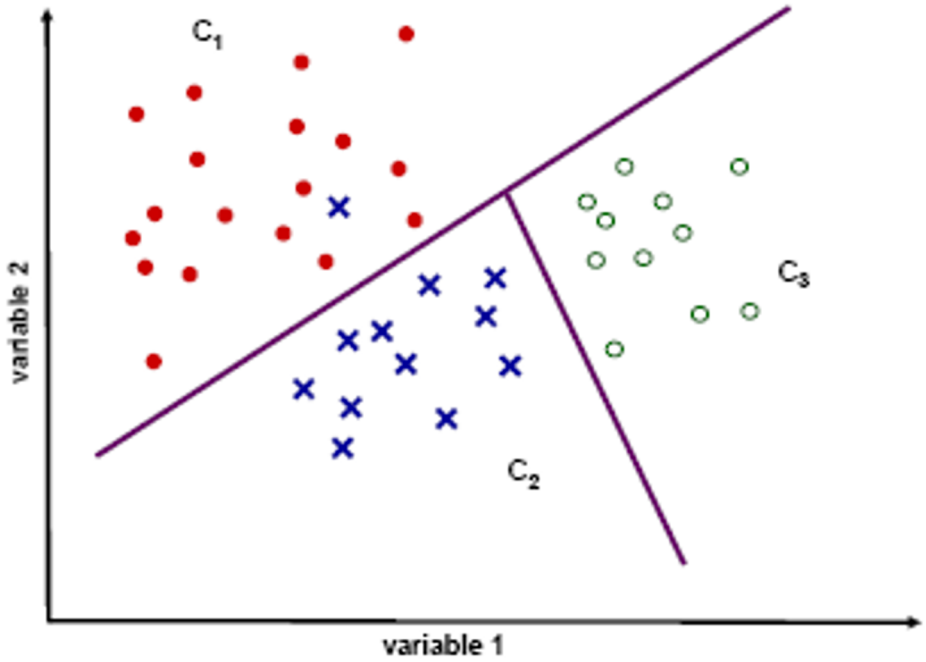
Statistical methods

Statistical methods

- Statistical methods
  - Aim: to minimize an estimate of the classifier's generalization error rate
  - Often for numeric attributes
    - Often, with assumptions about their distributions
  - Examples: discriminant analysis, SVM, (deep) neural networks...

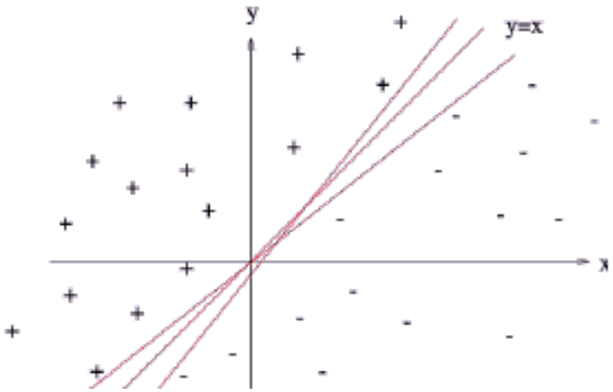
Linear Discriminant Analysis

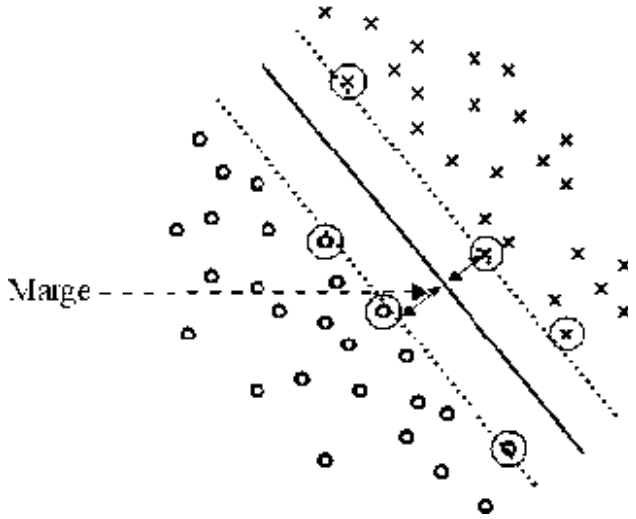
- Linear Discriminant Analysis (LDA)
  - Adapted to the case where the classes are linearly separable
  - Variants can handle non-linearly separable problems



Support Vector Machines

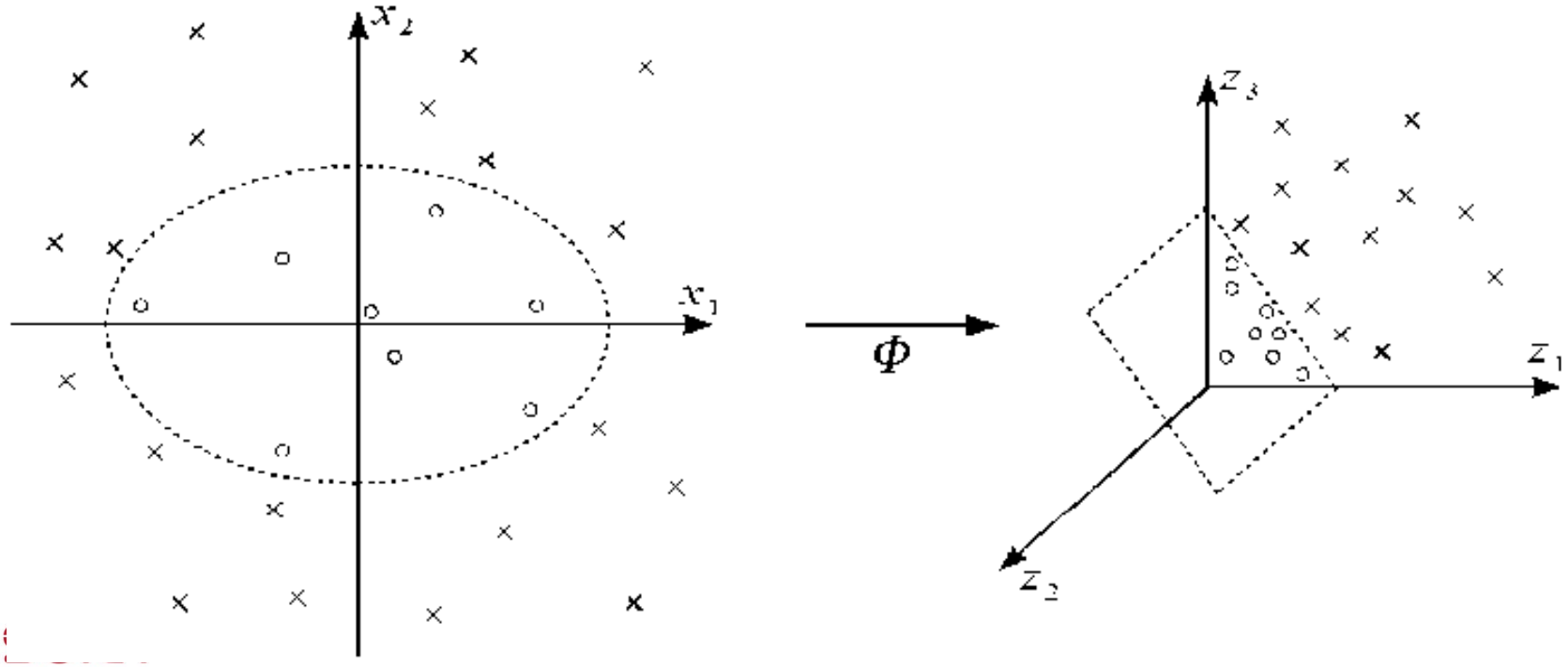
- Support Vector Machines (SVMs)
  - Can be applied only on numeric attributes
  - Initially, for linearly separable classes (can be extended)
  - Idea:
    - Maximizing the margin
    - Optimization under constraints





Support Vector Machines

- Non-linear extension of SVMs
  - Kernel « trick »
  - Can be applied to other linear methods, *e.g.* LDA



Support Vector Machines

- ☺ Each classification is associated with a confidence measure
  - As a function of the distance from margin
- ☺ Non-linear extensions are possibile
- ☺ Quick (to learn and to classify)
- ☺ Very good performance, especially in large dimensions
- ☺ Method that has been widely used in recent years

Support Vector Machines

☹ In general, can be applied only on **numeric** explanatory variables (and not to categorical variables)

☹ Very sensitive to dirty or ill-prepared data

- outliers
- redundant variables (might be interesting to apply PCA as a pre-processing)
- non-normalized data (attributes with different scales)

☹ Like most statistical models, SVMs are **black box** models

- Unlike symbolic models
- *(Not really in the trend of eXplainable AI…)*

Supervised classification

Performance evaluation

Assessing performance (1)

- *Theoretical evaluation:* study some theoretical properties of a method/model with some explicit mathematical proofs.
  - How many training instances are enough?
  - What is the expected accuracy of prediction?
  - Noise-resistance? …
- *Experimental evaluation:* observe the performance of a method in practical situations, using some datasets and a performance measure. Then make a summary from those experiments.
- We will discuss **only** experimental evaluation in this lecture

Assessing performance (2)

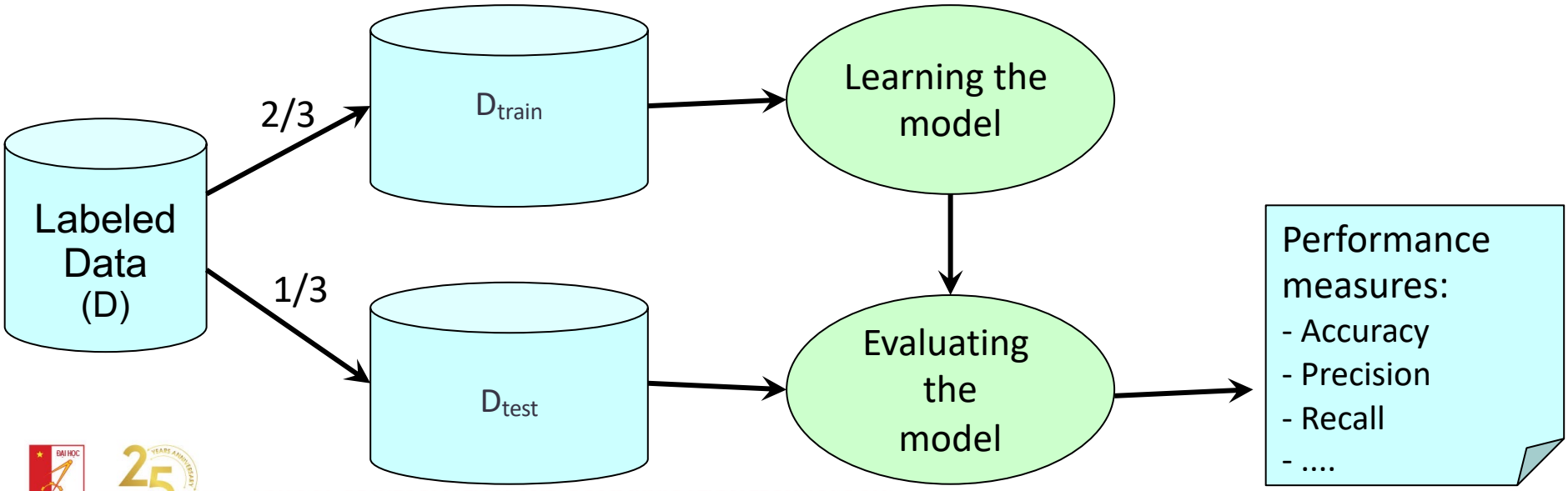
- ***Model assessment:*** *we need to evaluate the performance* of a method/model, only based on a given observed dataset D.
- Evaluation **strategies**:
  - To obtain a reliable assessment on performance.
- Evaluation **measures**:
  - To measure performance quantitatively.

Some evaluation strategies

- Hold-out
- Stratified sampling
- Repeated hold-out
- Cross-validation

Hold-out (random splitting)

- The observed, labeled dataset D is randomly split into 2 non-overlapping subsets:
  - D_{train}: used for training (learning dataset) -> the ground-truth (labels) is used by the model to learn correctly
  - D_{test}: used to test performance (test dataset) -> the ground-truth (labels) is only used to evaluate the performances of the model (after learning is completed)



Hold-out (random splitting)

- Note that:
  - No instance of D_{test} is used in the training phase.
  - No instance of D_{train} is used in the test phase.
- Popular split: $|D_{train}| = (2/3).|D|$, $|D_{test}| = (1/3).|D|$
- This technique is suitable when D is of large size.

Stratified sampling

- For small or imbalanced datasets, random splitting might result in a training dataset which are not representative.
  - A class in D_{train} might be empty or have few instances.
- *We should split D so that the class distribution in D_{train} is similar with that in D.*
- Stratified sampling fulfills this need:
  - *We randomly split each class of D into 2 parts:* one is for D_{train}, and the other is for D_{test}.
  - for each class: 

- Note that this technique cannot be applied to regression (nor unsupervised learning).

Repeated hold-out

- We can do hold-out many times, and then take the average result.
  - Repeat hold-out n times. The i^{th} time will give a performance result p_i. The training data for each hold-out should be different from each other.
  - Take the average $p = \text{mean}(p_1, \ldots, p_n)$ as the final quality.

Cross-validation

- In repeated hold-out: there are overlapping between two training/testing datasets. It might be redundant.
- *K-fold cross-validation:*
  - *Split D into K equal parts which are non-overlapping.*
  - *Do K runs (folds): at each run, one part is used for testing and the remaining parts are used for training.*
  - *Take the average as the final quality from K individual runs.*

- Popular choices of K: 10 or 5
- It is useful to combine this technique with stratified sampling.
- This technique is suitable for small/average datasets.

Model selection

- An ML method (including classifiers) often has a set of hyperparameters that require us to select suitable values
  - E.g. Tree: stopping criterion
  - E.g. Random forest: number of trees
- How to choose a good value?
- ***Model selection:*** *given a dataset D, we need to choose a good setting of the hyperparameters in method (model) A such that the classifier generalizes well.*
- A validation set T_{valid} is often used to find a good setting.
  - It is a subset of D.
  - A good setting should give good performance results on T_{valid}.

Model assessment and selection

- Given an observed dataset D, we need to **select** a good value for hyperparamerter λ and **evaluate** the overall performance of a method:
  - *Select a finite set S which contains all potential values of λ.*
  - *Select a performance measure P.*
  - *Split D into 3 non-overlapping subsets: D_{train}, T_{valid} and T_{test}*
  - *For each $\lambda \in S$: train the system given D_{train} and λ. Measure the quality on T_{valid} to get P_λ.*
  - *Select the best λ^* which corresponds to the best P_λ.*
  - Train the system again from $D_{train} \cup T_{valid}$ given λ^*.
  - Test performance of the system on T_{test}.
- Hold-out can be replaced with other techniques (*e.g.* cross-validation)

Recall: Assessing performance

- ***Model assessment:*** *we need to evaluate the performance* of a method/model, only based on a given observed dataset D.
- Evaluation **strategies**:
  - To obtain a reliable assessment on performance.
- Evaluation **measures**:
  - To measure performance quantitatively.
- So far, we presented only evaluation strategies….
  - Now, let's focus on evaluation (performance) measures!

Performance measures

- Effectiveness:
  - Ability to correctly predict the testing data.
- Efficiency:
  - The cost in time and storage when learning/prediction.
- Robustness:
  - The ability to reduce possible effects of noise/errors/missing data
- Scalability:
  - The relation between the performance and training size.
- Complexity:
  - The (time and space) complexity of the learned function.

Effectiveness measures: Accuracy

- Classification (often computed from the test dataset)

$$Accuracy = \frac{\#\ of\ well\text{-}classified\ records}{\text{total \# of records}}$$

Effectiveness measures: Precision-recall

- For each class c_i, we can compute the contingency matrix:

Predicted \ Real	Pos	Neg
Pos	TP	FP
Neg	FN	TN

- TP_i: # of records that are assigned correctly to c_i
- FP_i: # of records that are assigned incorrectly to c_i
- FN_i: # of records inside c_i that are assigned incorrectly to another class.
- TN_i: # of records outside c_i that are not assigned to class c_i.

- **Precision** for class c_i:
  - Percentage of correct instances, among all that are assigned to c_i.

$$Precision(c_i) = \frac{TP_i}{TP_i + FP_i}$$

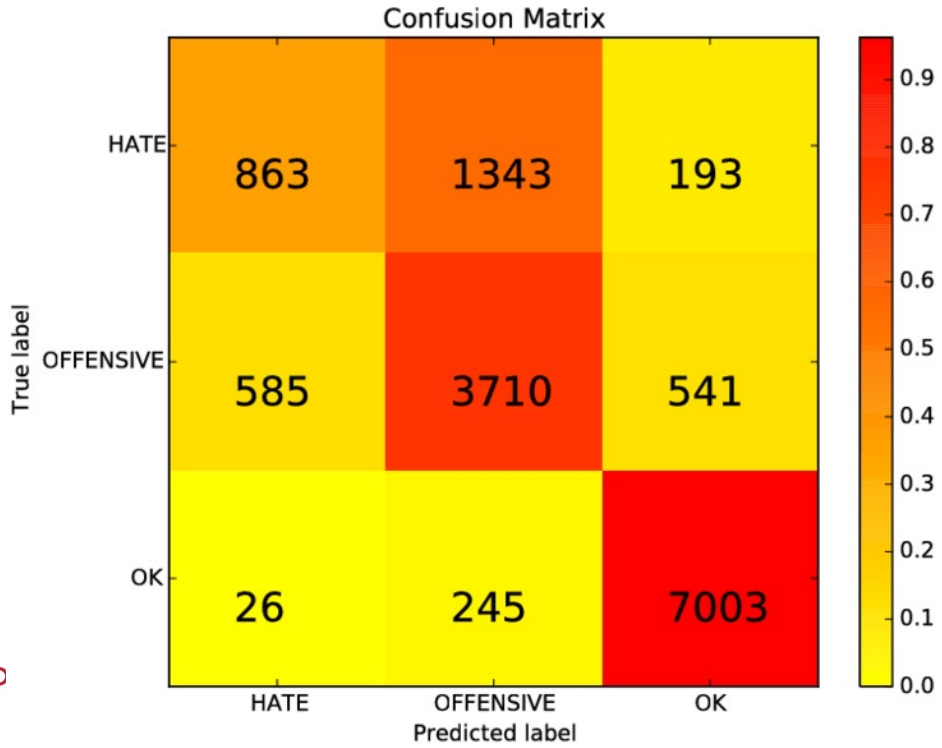
- **Recall** for class c_i:
  - Percentage of instances in c_i that are correctly assigned to c_i.

$$Recall(c_i) = \frac{TP_i}{TP_i + FN_i}$$

Effectiveness measures: Precision-recall - exercise

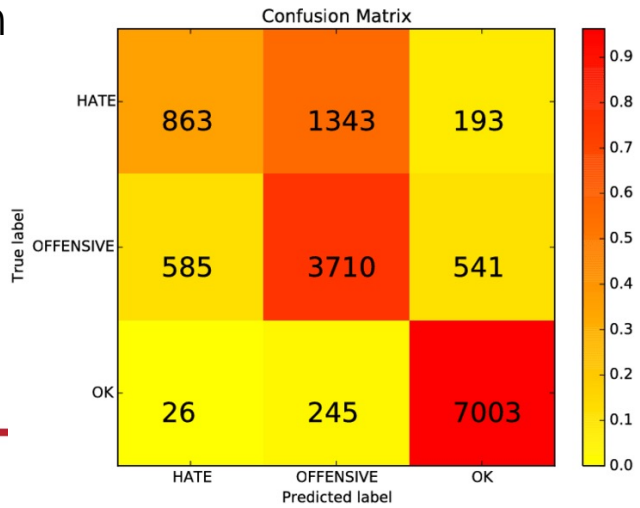
- Let's consider the following confusion matrix, with 3 classes
  - *(Application: detection of hate speech in texts on the web)*
  - Confusion matrix computed **from the test set**
  - Visualized as a heatmap for easier glimpse of the data

Predicted / true labels are inverted compared to the previous slide!



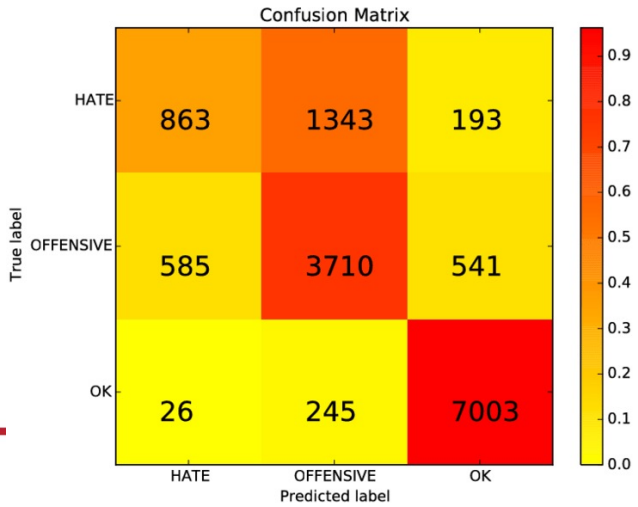
Effectiveness measures: Precision-recall - exercise

- Questions:
  - What is the number of TP for the class « offensive »?
    - Answer:
  - What is the number of FP for the class « offensive »
    - Answer:
  - What is the number of FN for the class « offensive »
    - Answer:
  - What is the precision for class « offensive »?
    - Answer:
  - What is the recall for class « offensive »?
    - Answer
  - When the system tells me that a given speech Is offensive, how much should I trust it?
    - Answer:
  - How good is the system in detecting offensive speeches?
    - Answer:



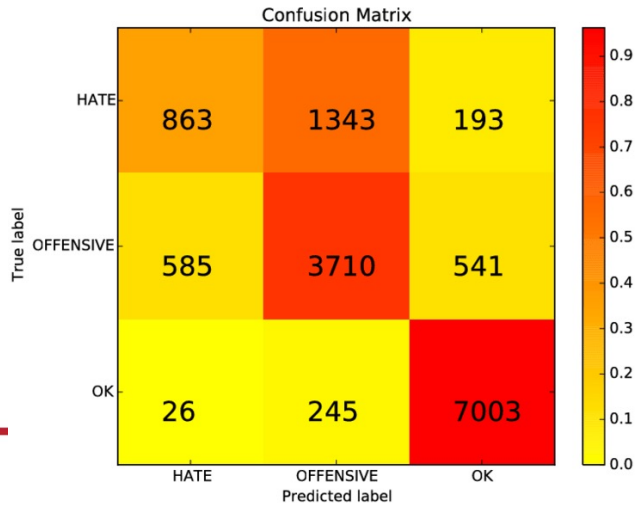
Effectiveness measures: Precision-recall - exercise

- Homework:
  - Same questions for the class « Hate »



Effectiveness measures: Precision-recall - exercise

- Comment: using such computations, one can compute the **average** precision, average recall, etc. over all classes
- In some applications, one might want to consider **weighted** average precisions
  - For example, if you want to make a system for protecting children from hate speech, will you give a higher or lower weight to the recall of the class « hate », compared to the class « OK »?
    - Answer:



Effectiveness measures: Sensitivity, specificity

- In medical applications, one commonly uses sensitivity, specificity
- **Sensitivity =** True Positive Rate = proportion of sick people that received a positive result on this test
- **Specificity** = True Negative Rate = proportion of healthy people that received a negative result on this test

sensitivity, recall, hit rate, or true positive rate (TPR)

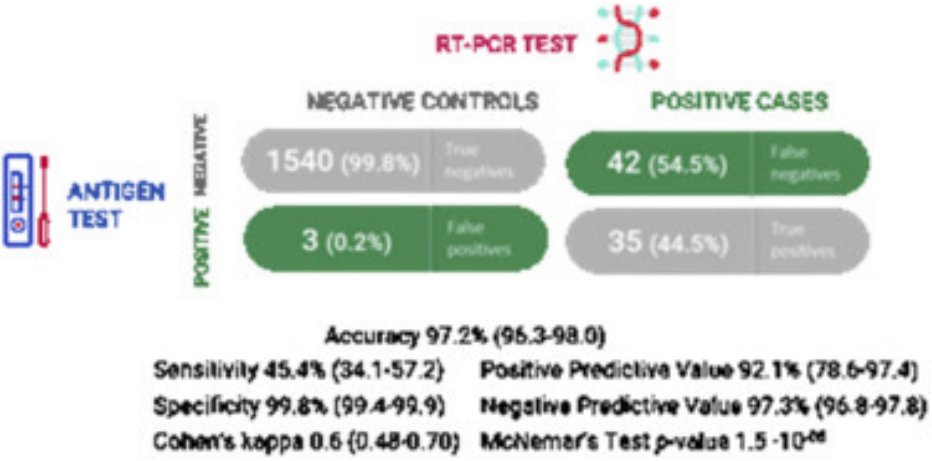
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

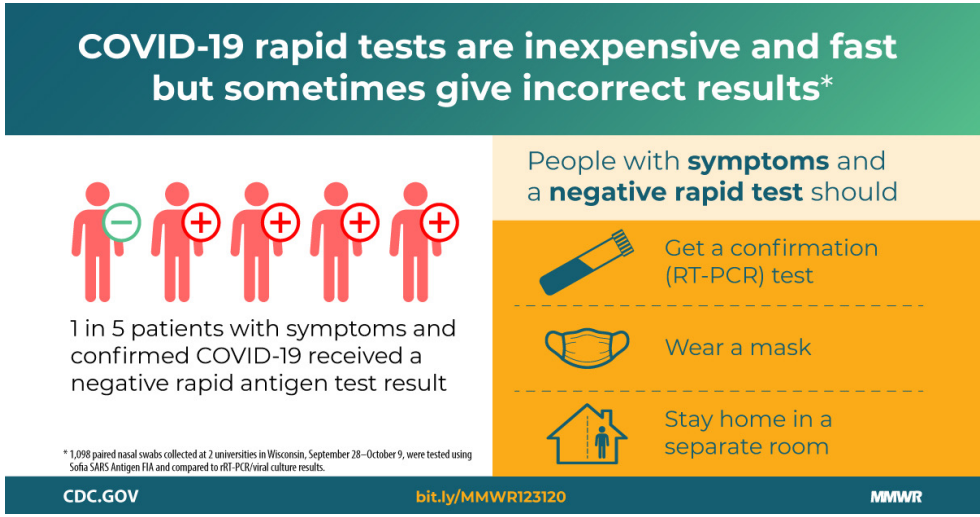
specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

Effectiveness measures: Sensitivity, specificity

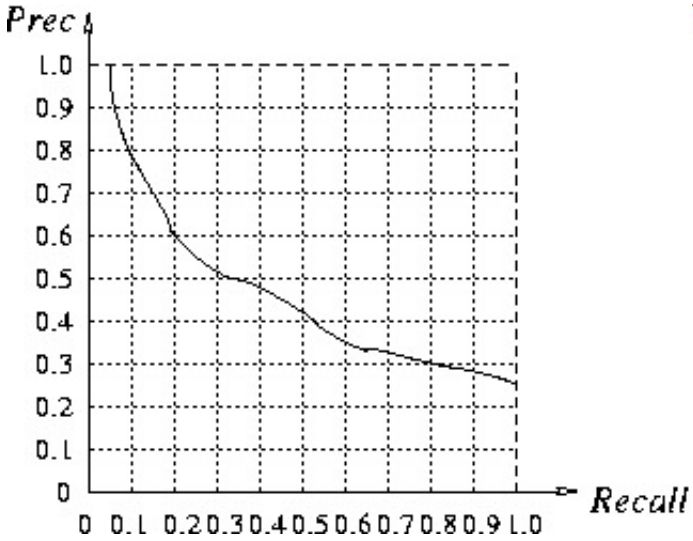
- Example with COVID tests: (no machine learning, but data science)
  - Sensitivity of (rapid) antigen test is lower than the sensitivity of PCR tests
    –> has more false negatives ☹
  - -> Need a policy (repeating tests, confirmation by PCR, prevention measures…)





Effectiveness measures: Precision-recall

- Precision-recall curves are **parametric** curves that can help us determine what are the best hyperparameters of our method: **Model selection**
  - It all depends on our objective!!!
    - In a biometric application (P=genuine person), having a high (recall or precision?) is more important than having a high (recall or precision?)
    - In COVID tests, having a high (recall or precision?) is more important than having a high (recall or precision?)



$$Precision(c_i) = \frac{TP_i}{TP_i + FP_i}$$

$$Recall(c_i) = \frac{TP_i}{TP_i + FN_i}$$

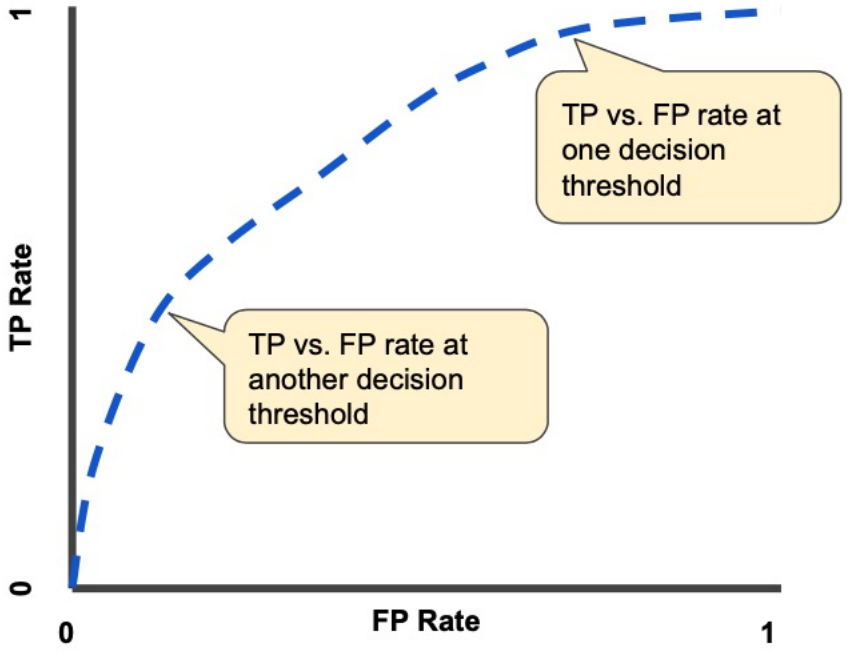
Effectiveness measures: F-measure

- Now that we have selected the best parameters for our model, we need to compare it with other methods
- F1-measure = Harmonic mean of the « best » precision and recall
  - Only applies where, given the application, there is no preference between recall and precision

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Effectiveness measures: ROC curve

- Now that we have selected the best parameters for our model, we need to compare it with other methods
- We vary the classification threshold and plot the ROC
- ROC curve (receiver operating characteristic curve)



True positive rate = recall

$$TPR = \frac{TP}{TP + FN}$$

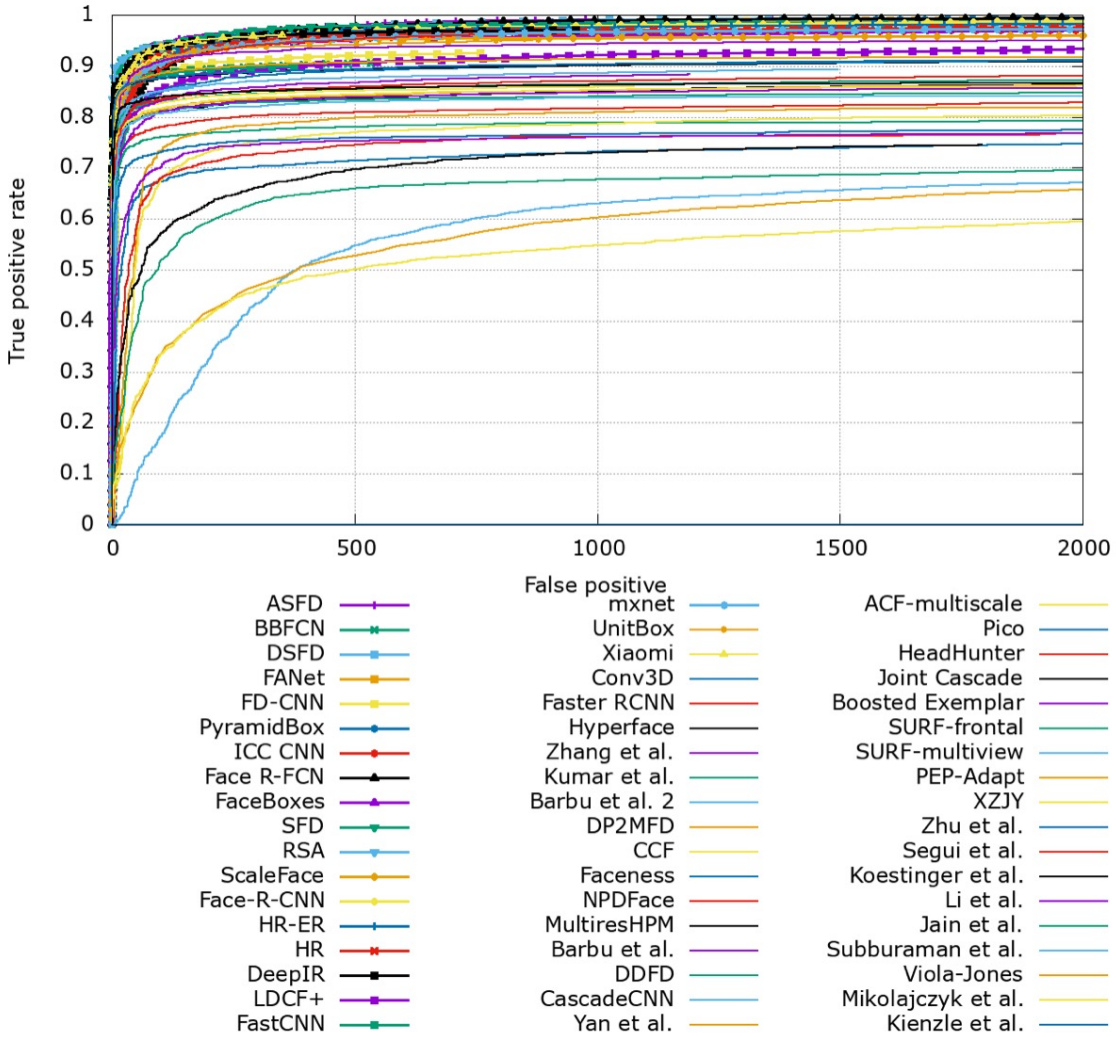
False Positive rate

$$FPR = \frac{FP}{FP + TN}$$

Effectiveness measures: ROC curve

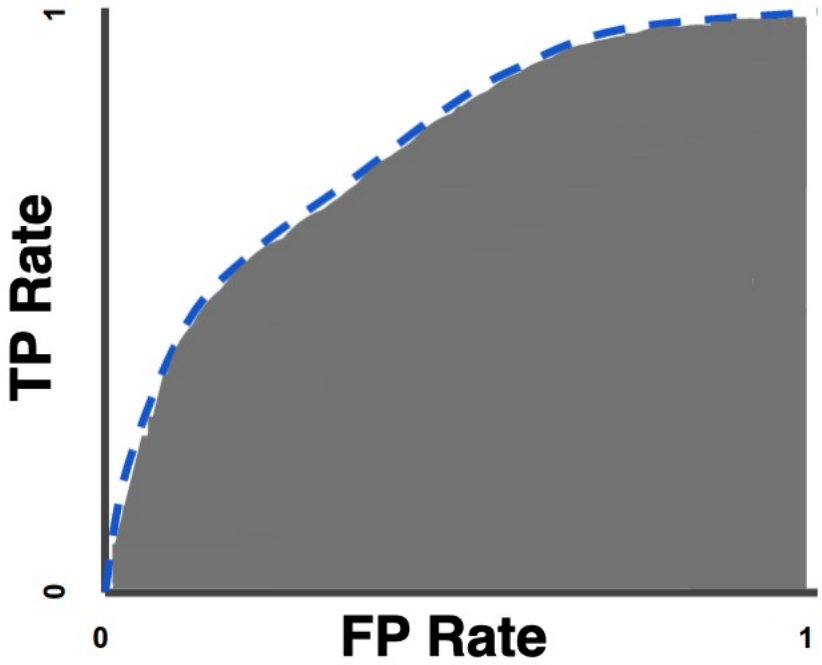
- Example with face detection algorithms

http://vis-www.cs.umass.edu/
fddb/results.html



Effectiveness measures: Precision-recall

- Area Under the Curve is a nice performance measure
  - **scalar** value (easy to compare different methods)
  - AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
  - AUC is classification-threshold-invariant



Which performance measure to use?

- Everything depends on the application, as explained here:
  - https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc

Short ID	learning_rate	n_estimators	roc_auc	avg_precision	f1_score	accuracy
BIN-103	0.05	3000	0.965006	0.871118	0.804844	0.96827
BIN-102	0.05	1500	0.964014	0.865733	0.79717	0.967437
BIN-101	0.1	1500	0.965926	0.873456	0.807781	0.968573
BIN-100	0.1	600	0.964205	0.864721	0.790019	0.966225
BIN-99	0.1	300	0.96103	0.845597	0.761358	0.962211
BIN-98	0.1	10	0.9198	0.694194	0.446406	0.93351
BIN-97	0.1	100	0.951547	0.806408	0.711948	0.956002

Performance measures

- Effectiveness:
  - Ability to correctly predict the testing data.
- Efficiency:
  - The cost in time and storage when learning/prediction.
- Robustness:
  - The ability to reduce possible effects of noise/errors/missing data
- Scalability:
  - The relation between the performance and training size.
- Complexity:
  - The (time and space) complexity of the learned function.

Performance measures

- Efficiency:
  - The cost in time and storage when learning/prediction.
- Not always possible to define it formally (depends on the data)
- **Time complexity**
  - For most applications, time complexity in the first step (learning) is not so important
    - Might be big, not really a problem for most applications
    - Example: we use GPUs to train deep learning models for computer vision
  - For most applications, we need low time complexity in the second step (prediction on new data)
    - Typically, needs to be performed at real-time or near real-time
  - Special case of **edge computing** -> definition of low-complexity models

Performance measures

- Efficiency:
  - The cost in time and storage when learning/prediction.
- Not always possible to define it formally (depends on the data)
- **Space complexity**
  - In some applications, it happens that learning cannot be completed because of space memory (RAM) needed > computer RAM
    - Example: AHC with big masses of data, ProDA in computer vision…
  - In those cases, we usually modify the method slightly so that our problem is tractable using the resources at-hand
  - Special case of **edge computing** -> definition of low-memory models

Chapter 5

Summary

Summary of Chapter 5

- Machine learning
  - Very « trendy » subfield of Artificial Intelligence
- Learning serves for **inductive inference**
  - Knowledge extraction from data (learning)
  - Performances are improved with experience (the more data, the better!)...
- Learning has several objectives
  - Prediction, segmentation, association, description…
  - ... that break down into ≠ tasks: dimensionality reduction, association rules, classification, regression, clustering...

Summary of chapter 5

- Learning can be **supervised** or not
- Different **unsupervised** methods for clustering:
  - Partition-based
  - Hierarchical
  - Grid-based methods
  - Density-based method
- Some parameters of the methods are important and can change the results, *e.g.* the distance / dissimilarity used
- Two types of performance measures:
  - Intrinsic measures: do not require any ground-truth
  - Extrinsic measures: based on comparison with some ground-truth
- There is no « best » method overall: everything depends on your data (volume, distribution, etc.), and your final objectives
  - For instance, some methods are better at dealing with outliers than others…

Summary of chapter 5

- Learning can be **supervised** or not
- Different **supervised** methods for classification:
  - Probabilistic methods
    - Pros: if D_{train} is very representative of the data, gives very good results
    - Cons: risk of over-fitting (bad generalization to new data)
  - Symbolic methods
    - Pros: very easily interpretable by a human
    - Cons: limited performance (especially with quantitative variables)
  - Statistical methods
    - Pros: excellent performance (especially on numeric data)
    - Cons: usually, lack of interpretability (black box models)
  - Many possible evaluation strategies and measures: depends mainly on the volume of data and your application
  - There is no « best » method overall: everything depends on your data (volume, distribution, etc.) and your final objective

References

- L. Breiman. *Random forests*. Machine learning, 45(1), 5-32, 2001.
- Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, Dinani Amorim. *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?* Journal of Machine Learning Research, 15(Oct):3133−3181, 2014.
- T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- M. Nunez. *The use of background knowledge in decision tree induction*. Machine Learning, 6(3): 231-250, 1991.
- Quinlan, J. R. *Induction of Decision Trees*. Mach. Learn. 1, 1 (Mar. 1986), 81-106, 1986
- Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.
- Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, *34*(1), 1-47.

Questions

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attention!!!