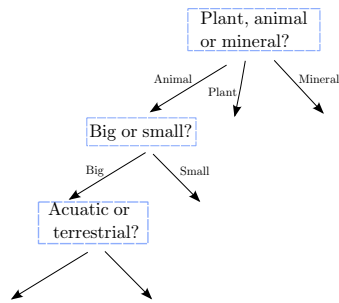


## Decision trees

- We can approach learning a concept as the algorithm to find the set of questions that is needed to distinguish it from others
- We can use a tree of questions as representation language, each node from the tree is a question about an attribute
- This representation is equivalent to a DNF ( $2^{2^n}$  concepts)
- Learning is performing a search in the space of trees of questions



Notes

---

---

---

---

---

---

---

---

## Decision trees

- To reduce the computational cost we impose a **bias** (the concepts that are preferred)
- **Constraint:** We want the tree that is the minimal description of the target concept given a set of examples
- **Justification:** This tree will be the best for classifying new examples (the probability of it having unnecessary conditions is reduced)
- **Occam's razor:** "Plurality ought never be posited without necessity"

Notes

---

---

---

---

---

---

---

---

## Decision trees Algorithms

- The first algorithm for decision trees was **ID3** (Quinlan 1986)
- It is a member of the family of algorithms for Top Down Induction Decision Trees (TDIDT)
- ID3 performs a Hill-Climbing search in the space of trees
- For each new question an attribute is chosen and the examples are partitioned according to their values, this procedure is repeated recursively with each partition until all examples are from the same concept
- The selection of the attribute is decided using an heuristic function that bias the selection towards the minimal tree

Notes

---

---

---

---

---

---

---

---

## Information theory

- Information theory studies (among other things) how to code messages and the cost of their transmission
- Given a set of messages  $M = \{m_1, m_2, \dots, m_n\}$ , each one with a probability of  $P(m_i)$ , we can define the quantity of information ( $I$ ) contained in a message from  $M$  as:

$$I(M) = \sum_{i=1}^n -P(m_i) \log(P(m_i))$$

- This value can be interpreted as the information necessary to distinguish among the messages from  $M$  (number of bits necessary to code the messages)

Notes

---

---

---

---

---

---

---

---

## Quantity of information as heuristic

- We can establish an analogy between learning and message coding supposing that the classes are the messages and the proportion of examples on each class are their probability (Learn a decision tree  $\Rightarrow$  Learn a code)
- A decision tree can be seen as the coding that allows to distinguish among classes
- We are looking for the minimal coding
- Each attribute must be evaluated to decide if it is included in the code
- For this heuristic an attribute is better if it can distinguish better among classes

Notes

---

---

---

---

---

---

---

---

## Quantity of information as heuristic

- At each level of the tree we look for the attribute that allows to minimize the code (reduces the size of the tree)
- This attribute is the one that minimizes the remaining quantity of information
- The selection of the attribute must result in a partition of examples where each subset is biased towards one of the classes
- We need an heuristic that measures the remaining information quantity induced by an attribute (Entropy,  $E$ )

Notes

---

---

---

---

---

---

---

---

## Quantity of information

- Given a set of examples  $\mathcal{X}$  and being  $\mathcal{C}$  their classification

$$I(\mathcal{X}, \mathcal{C}) = \sum_{\forall c_i \in \mathcal{C}} -\frac{\#c_i}{\#\mathcal{X}} \log\left(\frac{\#c_i}{\#\mathcal{X}}\right)$$

- Bits needed to code the examples without additional information

Notes

---

---

---

---

---

---

---

---

## Entropy

- Given an  $A$  attribute and being  $[A(x) = v_i]$  the examples with value  $v_i$  for the attribute

$$E(\mathcal{X}, A, \mathcal{C}) = \sum_{\forall v_i \in A} \frac{\#[A(x) = v_i]}{\#\mathcal{X}} I([A(x) = v_i], \mathcal{C})$$

- Bits needed to code the examples given an attribute

Notes

---

---

---

---

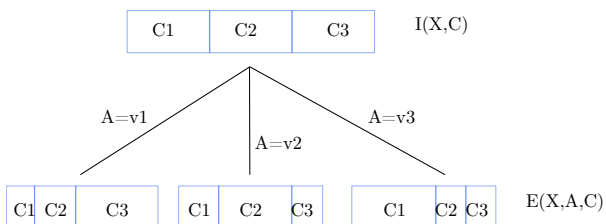
---

---

---

---

## Information gain



$$G(\mathcal{X}, A, \mathcal{C}) = I(\mathcal{X}, \mathcal{C}) - E(\mathcal{X}, A, \mathcal{C})$$

Notes

---

---

---

---

---

---

---

---

## ID3 algorithm

```

Algorithm: ID3 ( $\mathcal{X}$ : Examples,  $C$ : Classification,  $\mathcal{A}$ : Attributes)
if all examples are from the same class
then
  | return a leave with the class name
else
  | Compute the quantity of information of the examples (I)
  | foreach attribute in  $\mathcal{A}$  do
  |   | Compute the entropy (E) and the information gain (G)
  |   end
  |   Pick the attribute that maximizes G (a)
  |   Delete a from the list of attributes ( $\mathcal{A}$ )
  |   Generate a root node for the attribute a
  |   foreach partition generated by the values of the attribute a do
  |     |  $Tree_i = \text{ID3}(\text{examples from } \mathcal{X} \text{ with } \mathbf{a}=\mathbf{v}_i, \text{ examples classification, rest of attributes})$ 
  |     | generate a new branch with  $\mathbf{a}=\mathbf{v}_i$  and  $Tree_i$ 
  |   end
  | return the root node for a
end

```

Notes

---

---

---

---

---

---

---

---

---

---

## Algorithm ID3 - special cases

- The information gain is very small for all the attributes
  - It means that the remaining attributes are not discriminant
  - To continue adding decisions to the tree means to include arbitrary decisions
  - The best action is to stop and classify all the examples in the majority class
- A value from the attribute has no examples
  - Classify the value as the majority class
- Examples of different classes in a terminal node
  - Classify the value as the majority class

Notes

---

---

---

---

---

---

---

---

---

---

## Example (1)

Lets use the following set of examples

Ex.	Eyes	Hair	Height	Class
1	Blue	Blond	Tall	+
2	Blue	Black	Medium	+
3	Brown	Black	Medium	-
4	Green	Black	Medium	-
5	Green	Black	Tall	+
6	Brown	Black	Short	-
7	Green	Blond	Short	-
8	Blue	Black	Medium	+

Notes

---

---

---

---

---

---

---

---

---

---

## Example (2)

$$\begin{aligned}
 I(X, C) &= -1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2) = 1 \\
 E(X, \text{eyes}) &= (\text{blue}) 3/8 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\
 &\quad + (\text{brown}) 2/8 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\
 &\quad + (\text{green}) 3/8 \cdot (-1/3 \cdot \log(1/3) - 2/3 \cdot \log(2/3)) \\
 &= 0,344 \\
 E(X, \text{hair}) &= (\text{blond}) 2/8 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\
 &\quad + (\text{black}) 6/8 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\
 &= 1 \\
 E(X, \text{height}) &= (\text{tall}) 2/8 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\
 &\quad + (\text{medium}) 4/8 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\
 &\quad + (\text{short}) 2/8 \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \\
 &= 0,5
 \end{aligned}$$

Notes

---

---

---

---

---

---

---

---

## Example (3)

As we can see the attribute **eyes** is the one that maximizes the information gain

$$\begin{aligned}
 G(X, \text{eyes}) &= 1 - 0,344 = 0,656 * \\
 G(X, \text{hair}) &= 1 - 1 = 0 \\
 G(X, \text{height}) &= 1 - 0,5 = 0,5
 \end{aligned}$$

Notes

---

---

---

---

---

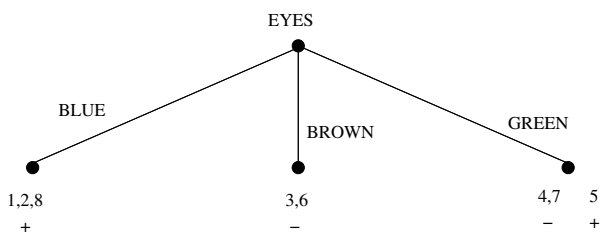
---

---

---

## Example (4)

This attributes induces a partition that forms the first level of the tree



Notes

---

---

---

---

---

---

---

---

## Example (5)

Now only in the node corresponding to the value **green** we have a mixture of examples from the two classes, we repeat the process with these examples.

Ej.	Hair	Height	Class
4	Black	Medium	-
5	Black	Tall	+
7	Blond	Short	-

Notes

---

---

---

---

---

---

---

---

## Example (6)

$$\begin{aligned}
 I(X, C) &= -1/3 \cdot \log(1/3) - 2/3 \cdot \log(2/3) = 0,918 \\
 E(X, hair) &= (blond) 1/3 \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \\
 &+ (black) 2/3 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\
 &= 0,666 \\
 E(X, height) &= (tall) 1/3 \cdot (0 \log(0) - 1 \cdot \log(1)) \\
 &+ (medium) 1/3 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\
 &+ (short) 1/3 \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \\
 &= 0
 \end{aligned}$$

Notes

---

---

---

---

---

---

---

---

## Example (7)

Now the attribute that maximizes the information gain is **eyes**.

$$\begin{aligned}
 G(X, hair) &= 0,918 - 0,666 = 0,252 \\
 G(X, height) &= 0,918 - 0 = 0,918*
 \end{aligned}$$

Notes

---

---

---

---

---

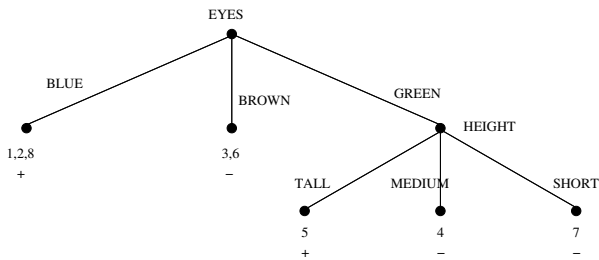
---

---

---

## Example (8)

Now the tree can discriminate all the examples



Notes

---

---

---

---

---

---

---

---

## Shortcomings of the information gain measure

- The information gain measure is biased towards the attributes that have more values
- **Problem:** If an attribute have more values probably the resulting tree will be larger
- The reason for that bias resides in the weight given to the values that are totally discriminant

Notes

---

---

---

---

---

---

---

---

## Shortcomings of the information gain measure - Example

A <sub>1</sub>	n. exam	A <sub>2</sub>	n. ejem
a	20+	a	40+
b	25+	b	40-
c	20-	c	10+ 10-
d	15-		
e	5+ 15-		

$$G(A_1) = 1 - \left( \frac{1}{5} \cdot 0 + \frac{1}{4} \cdot 0 + \frac{1}{5} \cdot 0 + \frac{3}{20} \cdot 0 + \frac{1}{5} \cdot \left( -\frac{1}{4} \cdot \log \frac{1}{4} - \frac{3}{4} \cdot \log \frac{3}{4} \right) \right) = 0,84$$

$$G(A_2) = 1 - \left( \frac{2}{5} \cdot 0 + \frac{2}{5} \cdot 0 + \frac{1}{5} \cdot \left( -\frac{1}{2} \cdot \log \frac{1}{2} - \frac{1}{2} \cdot \log \frac{1}{2} \right) \right) = 0,8$$

Notes

---

---

---

---

---

---

---

---

## A normalized value

- This problem can be reduced by normalizing the information gain function with another that measures the distribution of the examples in an attribute

$$SI(\mathcal{X}, A) = - \sum_{\forall v_i \in A} \frac{\#[A(x) = v_i]}{\#\mathcal{X}} \cdot \log\left(\frac{\#[A(x) = v_i]}{\#\mathcal{X}}\right)$$

- for the example:

$$\frac{G(A_1)}{SI(A_1)} = \frac{0,86}{2,3} = 0,36 \quad \frac{G(A_2)}{SI(A_2)} = \frac{0,8}{1,52} = 0,52$$

Notes

---

---

---

---

---

---

---

---

## Numerical Attributes

- The heuristic functions defined are only able to treat discrete attributes
- Usually data sets are also described by numerical attributes
- To treat this attributes as discrete attributes is unfeasible (each value a different value)
- Solution:** Partition the range of values in two or more intervals
- In practice this means to discretize the attribute in two or more values

Notes

---

---

---

---

---

---

---

---

## Discretization of numerical attributes

- We begin with the ordered sequence of values  $A_i = \{v_1, v_2, \dots, v_n\}$
- The information gain is measure for each partition of the sequence  $(n - 1)$
- The partition that maximizes the information gain is then compared with the rest of the attributes
- The decision in the tree will be  $[A[x] < v_i]$  and  $[A[x] \geq v_i]$
- We can not discard the attribute from successive decision (we can partition further the intervals)
- This will increment the computational cost

Notes

---

---

---

---

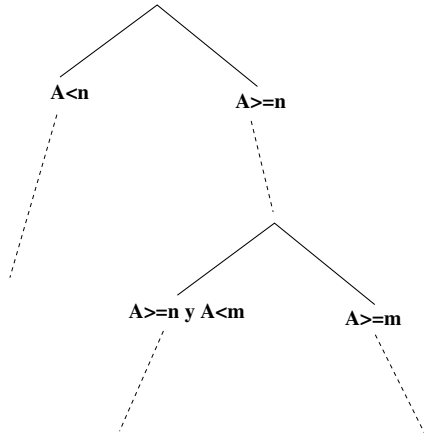
---

---

---

---

## Numerical attributes - Node decisions



Notes

---

---

---

---

---

---

---

---

## Distribution of numerical attributes

1	2	3	4	5	6	7	8	9	10
+	+	+	+	+	-	-	-	-	-
	.89	.76	.6	.39	<b>0</b>	.39	.6	.76	.89
+	+	+	+	-	+	-	-	-	-
	.89	.76	.6	<b>.39</b>	.72	<b>.39</b>	.6	.76	.89
+	+	-	-	-	+	+	+	-	-
	.89	<b>.76</b>	.96	1	.97	1	.96	<b>.76</b>	.89
+	+	-	+	+	+	+	-	-	+
	.82	.76	.87	.87	.84	.79	<b>.68</b>	.84	.82

Notes

---

---

---

---

---

---

---

---

## Missing values

- A **missing value** is a value for the attribute of an example that is unknown
- In practice real data sets have this kind of values
- We can find problems processing these values when:
  - The best attribute is picked (how compute the information gain)
  - The branches of an attribute are created (where to put the examples)
  - When predicting the class of a new example (what branch to follow)

Notes

---

---

---

---

---

---

---

---

## Missing values - Solutions

Notes

## When picking the best attribute

- 1 Ignore the examples
- 2 Scale the information gain

$$G'(\mathcal{X}, A, \mathcal{C}) = \frac{\#[A(x) \neq v_{miss}]}{\#\mathcal{X}} \cdot G(\mathcal{X}, A, \mathcal{C})$$

- 3 Substitute the values (mode / median)

## Missing values - Solutions

Notes

## Creating the branches

- 1 Ignore the examples
- 2 Use the most frequent value
- 3 Assign a fraction of the example to each branch proportionally to the distribution of the values of the attribute.
- 4 Include the instance on each branch
- 5 Add a new branch for the missing values

## Missing values - Solutions

Notes

## Predicting a new example

- 1 Use the branch for missing values if there is one
- 2 Use the most frequent value
- 3 Explore all the branches and return a probability for each class
- 4 Return the majority class

## Overcoming noise

- **Noise:** Examples that are wrongly classified
- It is usual for real datasets that a part of the examples are misclassified
- The effect of noisy datasets is that the size of the learned trees is larger
- Extra nodes appear to distinguish the misclassified examples
- This effect is called **overfitting**
- Overfitting reduces the generality of the tree and as consequence the prediction accuracy is reduced

Notes

---

---

---

---

---

---

---

---

## Pruning of decision trees

- A way to reduce the effect of noise is to delete the subtrees that incorrectly classify examples
- These techniques are called **pruning**
- The idea is to find the nodes that if deleted from the tree improve the global accuracy
- Two possibilities:
  - Pre-pruning: prune while the tree is built
  - Post-pruning: prune after the tree is completed

Notes

---

---

---

---

---

---

---

---

## Pre-pruning

- It is based on statistical tests that determine the benefits of creating a new decision
- The best accuracy of the best decision for a node is compared with the accuracy obtained by stopping creating new decisions
- Test for comparing probability distributions:

$$\sum_{c_i \in \mathcal{C}} \sum_{v_i \in A} \frac{(P(c_i|A(x) = v_i) - P(c_i))^2}{P(c_i)}$$

distributes as a  $\chi^2$  with  $(c - 1) \cdot (v - 1)$  degrees of freedom

Notes

---

---

---

---

---

---

---

---

## Post-pruning

- The prediction error can be estimated with the examples from dataset used to learn the tree
- Being  $N$  the number of examples in a node,  $E$  the number of examples that are on the majority class and  $B$  the binomial probability distribution function, the estimated error for a node can be computed as:

$$Error = N \cdot B_{cf}(E, N)$$

- Where  $cf$  is the confidence factor of the binomial probability function (usually 0.25)
- If the error of the node is less than the weighted sum of the errors of its descendant then they can be pruned

Notes

---

---

---

---

---

---

---

---

## Post-pruning - Example (1)

Example	Price	Size	Country	Motor	Class
1	expensive	big	france	diesel	compact
2	cheap	medium	germany	diesel	compact
3	medium	medium	germany	gasoline	compact
4	cheap	big	france	gasoline	compact
5	cheap	big	france	gasoline	compact
6	expensive	small	japan	gasoline	sports
7	expensive	small	germany	gasoline	sports
8	expensive	small	USA	gasoline	sports
9	medium	small	japan	gasoline	sports
10	medium	small	germany	gasoline	sports
11	expensive	big	USA	gasoline	luxury
12	expensive	big	USA	diesel	luxury
13	expensive	big	france	gasoline	luxury
14	expensive	big	germany	gasoline	luxury
15	expensive	big	germany	diesel	luxury

Notes

---

---

---

---

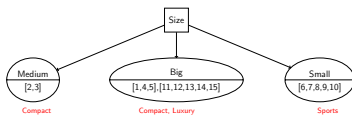
---

---

---

---

## Post-pruning - Example (2)



Notes

---

---

---

---

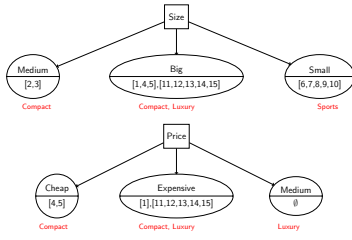
---

---

---

---

## Post-pruning - Example (2)



Notes

---

---

---

---

---

---

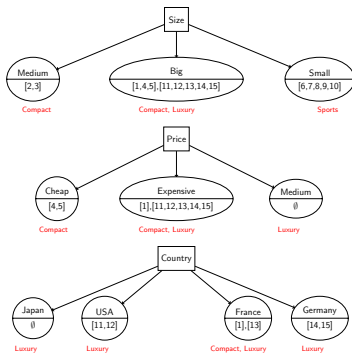
---

---

---

---

## Post-pruning - Example (2)



Notes

---

---

---

---

---

---

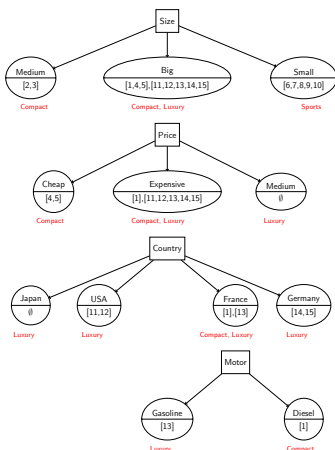
---

---

---

---

## Post-pruning - Example (2)



Notes

---

---

---

---

---

---

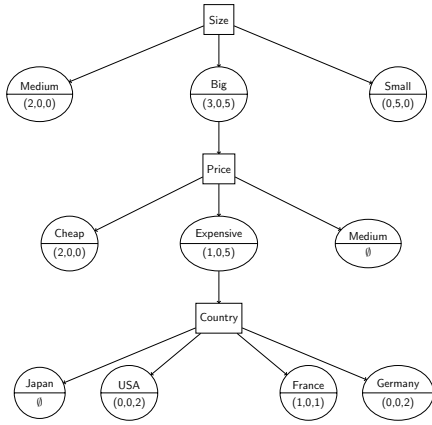
---

---

---

---

### Post-pruning - Example (3)



Notes

---

---

---

---

---

---

---

---

### Post-pruning - Example (4)

- Node Expensive

$$Error_{expensive} = 6 \cdot B_{0,25}(1, 6) = 6 \cdot 0,5339 = 3,203$$

$$Error_{\{USA, Ger, Fr\}} = 2 \cdot B_{0,25}(0, 2) + 2 \cdot B_{0,25}(0, 2) + 2 \cdot B_{0,25}(1, 2) = 4,125$$

- Node Big

$$Error_{big} = 8 \cdot B_{0,25}(3, 8) = 8 \cdot 0,8862 = 7,089$$

$$Error_{\{cheap, expensive\}} = 2 \cdot B_{0,25}(0, 2) + 6 \cdot B_{0,25}(1, 6) = 4,327$$

Notes

---

---

---

---

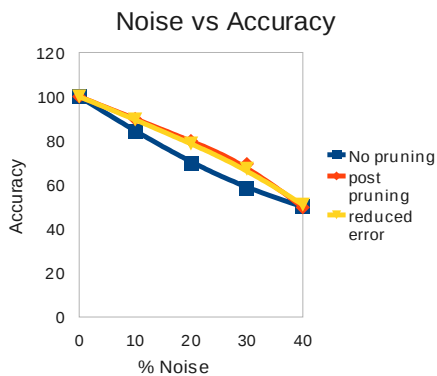
---

---

---

---

### Pruning / Noise and % accuracy



Notes

---

---

---

---

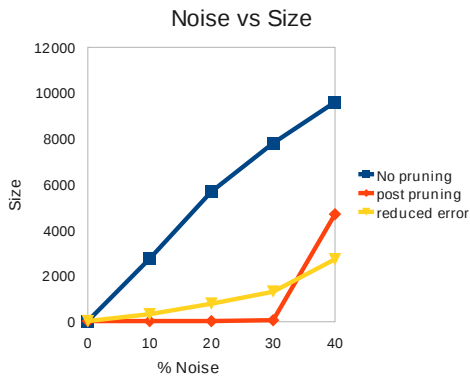
---

---

---

---

## Pruning / Nose and tree size



Notes

---

---

---

---

---

---

---

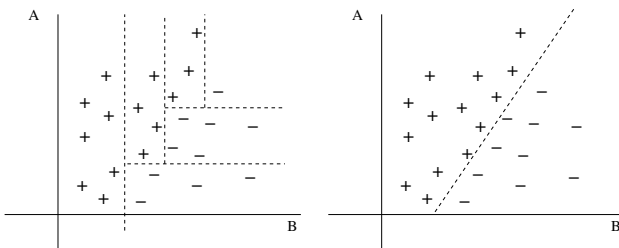
---

---

---

## Limitations of ID3

- ID3 can learn any concept, but there are concepts that are more difficult to learn than others
- The main problem is that the representation uses partitions that are parallel to the coordinate axis
- To approximate this way some concepts could result in very complex trees



Notes

---

---

---

---

---

---

---

---

---

---

## Other algorithms for decision trees

- CART (Classification And Regression Trees)
  - GINI index  $G(X, C) = 1 - \sum P(X|C)^2$
  - Linear combination of attributes (oblique partitions)
- **Regression trees:** Decision trees for predicting continuous values (functions)
- **Model trees:** Decision trees where the leaves are a model obtained by other algorithm

Notes

---

---

---

---

---

---

---

---

---

---

## Rule induction

- Decision rules can be generated from a decision tree
- Each path from the root of the tree to a leaf is a rule that classifies a set of examples
- It is also possible to generate a set of rules without creating a decision tree
- These algorithms learn the rules sequentially and not all at one like in a decision tree
- Some of these algorithm can learn first order rules

Notes

---



---



---



---



---



---



---

## Sequential covering algorithms

- These algorithms generate the rules sequentially by looking for the best rule that covers a subset of the examples
- We look for the rule with the best accuracy but not the maximum coverage
- Once we have the best rule, we can eliminate the examples covered
- This procedure can be iterated until we have rules that cover all the dataset
- The result is a disjunction of rules that can be ordered by accuracy

Notes

---



---



---



---



---



---



---

## Sequential covering algorithms - Algorithm

---



---

```

Algorithm: sequential-covering(attributes,examples,min)
list_rules=[]
rule = learn-one-rule(attributes,examples)
while accuracy(rule,examples)>min do
  list_rules= list_rules + rule
  examples = examples - correct-examples(rule, examples)
  rule = learn-one-rule(attributes,examples)
end
list_rules = order-by-accuracy(list_rules)

```

---

Notes

---



---



---



---



---



---



---

## Learning one rule

- The key point of these methods is how to learn the best rule given a set of examples
- One possibility is to take the idea from decision trees looking in the space of conjunctions
- We start with the empty rule and each step we choose the best condition for the rule using an heuristic function (for instance entropy) and a greedy strategy
- To avoid local optimum a more exhaustive search can be performed, for example performing a beam search that stores the  $k$  best rules
- The rule is assigned as a predictor for the majority class of the examples selected

Notes

---

---

---

---

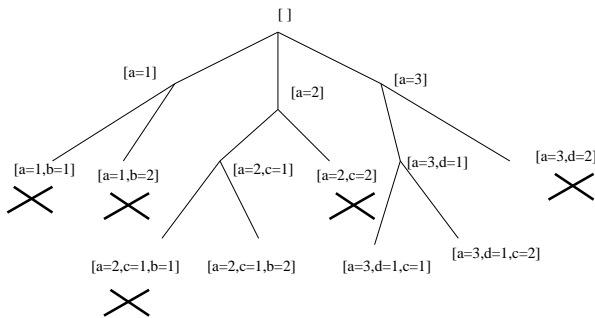
---

---

---

---

## Example



Notes

---

---

---

---

---

---

---

---

## Other algorithms for rule induction

- The sequential method has its limitations, there exists a large variety of alternatives
- It can be changed how the instances are selected for covering (for example focusing on the classes)
- Other formalism of rules can be used (Decision tables, ripple down rules, default rules, ...)
- Inductive Logic Programming (ILP)
  - These algorithms can be extended for learning first order rules (rules with variable, recursive rules, ...)
  - These kind of rules allow to learn more complex concepts (relational knowledge, sub structures, programs, ...)

Notes

---

---

---

---

---

---

---

---

## Classifier ensembles/Meta learning

- An alternative to improve the accuracy of a classifier is to combine the decision of a set of classifiers (*ensembles*)
- An ensemble of simple classifiers trained using different datasets can obtain better accuracy than a complex classifier
- Each classifier has a different “*point of view*” of the concept, their combination gives a more complete vision
- Each classifier gives a prediction, all results are combined to obtain the final prediction (majority, weighted vote, ...)
- This methodology works well usually when the learning algorithm has large variability (weak classifiers)

Notes

---

---

---

---

---

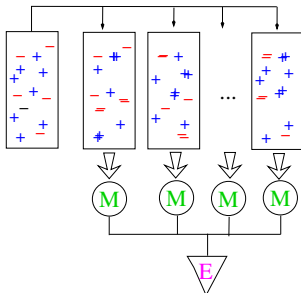
---

---

---

## Classifier ensembles - Bagging

- **Bagging (Bootstrapping aggregation):**  $n$  datasets are generated from an original dataset using extraction with replacement sampling. All classifier are trained the same way



Notes

---

---

---

---

---

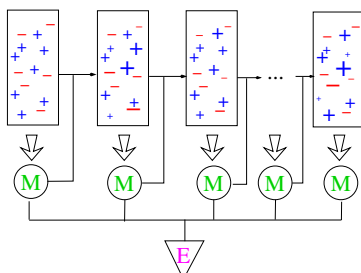
---

---

---

## Classifier ensembles - Boosting

- **Boosting:** Each instance in the dataset has a weight. Successive classifiers are generated increasing the weight of the instances that are not predicted correctly and decreasing the weight of the instances that are correctly predicted. Each classifier specializes on the difficult instances for the previous classifier.



Notes

---

---

---

---

---

---

---

---

## Classifier ensembles - Random Subspaces

- **Random Subspaces:** We actually do not know what attributes are relevant for classifying the examples. We can select a subset of the attributes and obtain a more simple view of the task. The same classifier trained in different subspaces can be combined to obtain a global view from different perspectives
- **Example:** Random forest, a combination of decision trees trained with subsets of attributes.

Notes

---



---



---



---



---



---



---

## Classifier ensembles - Other approaches

- **Voting:** Build N different classifiers (different algorithms) and use majority vote to obtain the result
- **Stacking:** Build N different classifiers (different algorithms) to generate N new attributes and learn to classify the augmented dataset with other classifier
- **Classification by regression:** Build N binary regression classifiers, one for each class, for each classifier assign +1 to the target class and -1 to the rest, The prediction is the classifier that obtains the higher positive value for its class.

Notes

---



---



---



---



---



---



---

## K-nearest neighbours

- *Lazy learning* mechanism
- There is not a model associated to the learned concepts
- Predictions are obtained from the training examples that are more similar to the one that is being classified
- The cost of the learning process is 0, all the cost is in the computation of the prediction
- A distance function to compare the examples similarity is needed (for instance euclidean distance)
- This means that if we change the distance function, we change how examples are classified

Notes

---



---



---



---



---

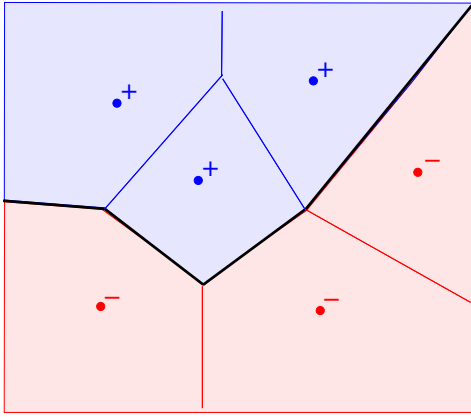


---



---

## K-nearest neighbours - hypothesis space (1 neighbour)



Notes

---

---

---

---

---

---

---

---

## K-nearest neighbours - Algorithm

- Training: Store all the examples
- Prediction:
  - Let be  $x_1, \dots, x_k$  the  $k$  more similar examples to the one to classify
  - $\text{class}(x_{new}) = \text{combine\_predictions}(x_1, \dots, x_k)$
- The parameters of the algorithm are the number  $k$  of neighbours and the procedure for combining the predictions of the  $k$  examples

Notes

---

---

---

---

---

---

---

---

## K-nearest neighbours - Variants

- There are different possibilities for computing the class from the  $k$  nearest neighbours
  - Majority vote
  - Distance weighted vote
    - Inverse of the distance
    - Inverse of the square of the distance
    - Kernel functions (gaussian kernel, tricube kernel, ...)
- The model can be improved using the performance of the classifier (number of correct/incorrect predictions)
  - Weighting the examples
  - Weighting the attributes

Notes

---

---

---

---

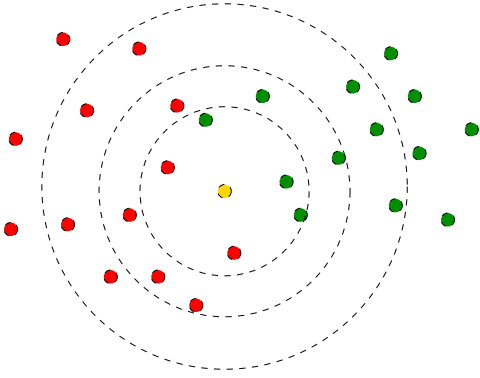
---

---

---

---

## K-nearest neighbours - prediction



Notes

---

---

---

---

---

---

---

---

## K-nearest neighbours - Advantages

- The cost of the learning process is zero
- No assumptions about the characteristics of the concepts to learn have to be done
- Complex concepts can be learned by local approximation using simple procedures
- The mechanism can be extended to predict a continuous values (regression, *Locally Weighted regression*)

Notes

---

---

---

---

---

---

---

---

## K-nearest neighbours - Drawbacks

- It is computationally expensive to find the  $k$  nearest neighbours (specialized data structure for N-dimensional indexing *kd-trees*)
- There is no mechanism for deciding the optimal value for  $k$  (depends on the dataset)
- Accuracy depends on the number of attributes of the dataset (*curse of dimensionality*)
- The model can not be interpreted (there is no description of the learned concepts)

Notes

---

---

---

---

---

---

---

---

## Bayesian learning

- We suppose that concepts are represented by n-dimensional probability distributions
- By the Bayes theorem

$$P(C_i|x) = \frac{P(C_i) \cdot P(x|C_i)}{P(x)}$$

- The prediction for an instance is the class with the larger probability
- The goal is to estimate the **probability density** function (PDF) of the classes
- To estimate a PDF we have to make some assumptions about the model of the distribution of the attributes of the dataset and their interactions

Notes

---



---



---



---



---



---



---

## Bayesian learning - Naive Bayes

- The simple approach is to assume that all the variables are independent (not usually true)
- The PDF can be transformed in the product of  $n$  PDFs

$$P(C_i|x) = \frac{P(C_i) \cdot \prod_{\forall a \in A} P(a(x) = v|C_i)}{\prod_{\forall a \in A} P(a(x) = v)}$$

- $\prod_{\forall a \in A} P(a(x) = v)$  is independent of the class and can be eliminated
- We estimate separately each  $P(a(x) = v|C_i)$  from the data
- Surprisingly this approach many times works better than more complex methods

Notes

---



---



---



---



---



---



---

## Bayesian learning - Probability estimation (NB)

- For discrete attributes we can estimate  $P(a(x) = v|C_i)$  as the frequency of the value of the attribute in the dataset (multinomial distribution)
- For continuous attributes we can estimate  $P(a(x) = v|C_i)$  assuming that they follow a continuous distribution (for instance gaussian) and estimate its parameters from the dataset
- During the estimation of the distributions we have to face the problem of having a small sample (the estimated probability is 0 for certain values or classes)
  - For discrete data ( $n_c$  frequency of the value in the class,  $n$  number of examples,  $p$  a priori probability (for instance uniformly distributed),  $m$  weighting constant) (**LaPlace correction**)

$$P(a(x) = v|C_i) = \frac{n_c + m \cdot p}{n + m}$$

Notes

---



---



---



---



---

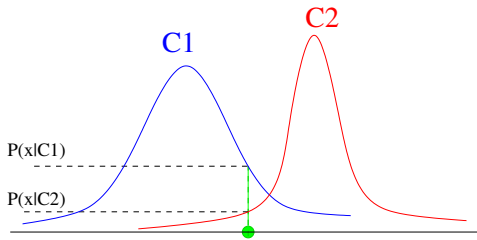


---



---

## Bayesian learning - Prediction (NB)



$$P(C1|x) = \frac{P(x|C1)}{P(x|C1) + P(x|C2)}$$

Notes

---

---

---

---

---

---

---

---

## Bayesian learning - Advantages and drawbacks (NB)

- **Advantages:**
  - It is simple and computationally cheap
  - Its result can be good even when its assumptions are not true
- **Drawbacks:**
  - The dataset is not correctly characterized (we are assuming things that are not true)
  - We have to decide what probability distribution represent the attributes

Notes

---

---

---

---

---

---

---

---

## Result validation

- To assess the quality of a classifier is a difficult task
- The quality of the classifier depends on the size of the training dataset and the quality of the data (noise, irrelevant attributes, missing values, ...)
- A simple assessment method when we have a large dataset is to use a part for training (*training set*) and the rest as testing (*test set*)
- Unfortunately this method usually gives optimistic results (the reported accuracy is better than in reality)
- Many classifiers have large variability depending on the training data (the same classifier trained with slightly different training sets can give very different results)

Notes

---

---

---

---

---

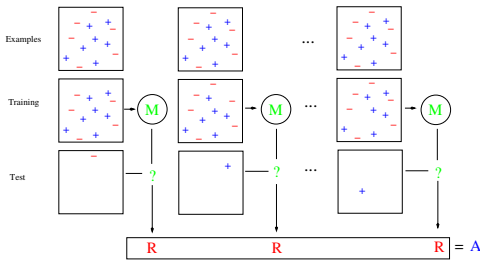
---

---

---

## Result validation

- **Leave-one-out:** The following procedure is repeated  $n$  times, use all the examples but one as training set and predict the one that has not been used (usefull if the dataset is small)



Notes

---

---

---

---

---

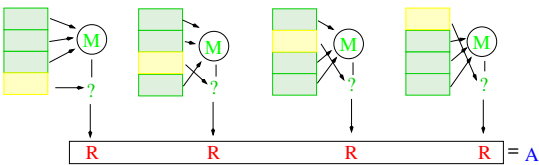
---

---

---

## Result validation

- **K-fold cross validation, stratified K-fold cross validation:** the dataset is divided in  $n$  subsets of the same size, the classifier is trained with  $n - 1$  subsets and evaluated with the one left aside (the stratified evaluation samples the dataset so each subset has the same class distribution as the whole dataset)



Notes

---

---

---

---

---

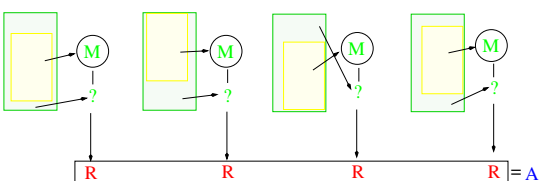
---

---

---

## Result validation

- **Bootstrapping:** Using extraction with replacement from the original dataset  $m$  datasets are generated and a percentage of each dataset is used as training set and the rest as test set



Notes

---

---

---

---

---

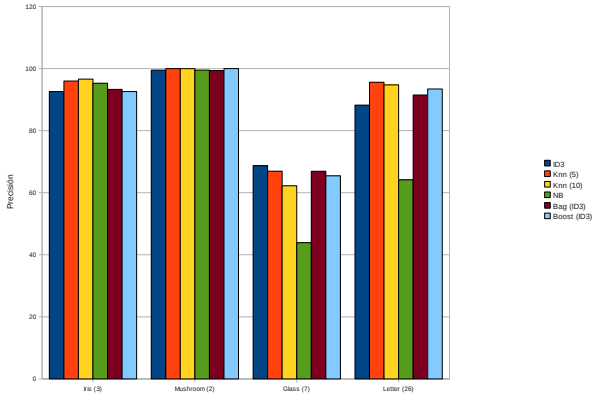
---

---

---

# Algorithm comparison

Notes



Notes

Notes