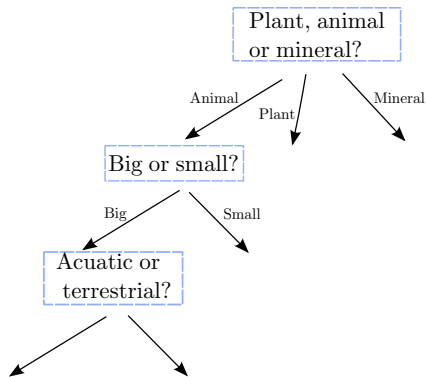


Decision trees

- 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



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"*

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 accordingly 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

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**)

Quantity of information as classification 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

Quantity of information as classification 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)

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

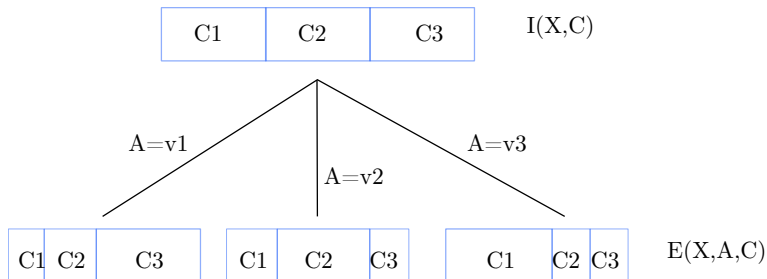
Entropy

- 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

Information gain



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

ID3 algorithm

Algorithm: ID3 (\mathcal{X} : Examples, \mathcal{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

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

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	+

Example (2)

$$I(X, C) = -1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2) = 1$$

$$\begin{aligned} E(X, \text{eyes}) &= (\text{blue}) 3/8 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\ &+ (\text{brown}) 2/8 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\ &+ (\text{green}) 3/8 \cdot (-1/3 \cdot \log(1/3) - 2/3 \cdot \log(2/3)) \\ &= 0,344 \end{aligned}$$

$$\begin{aligned} E(X, \text{hair}) &= (\text{blond}) 2/8 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\ &+ (\text{black}) 6/8 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\ &= 1 \end{aligned}$$

$$\begin{aligned} E(X, \text{height}) &= (\text{tall}) 2/8 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\ &+ (\text{medium}) 4/8 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\ &+ (\text{short}) 2/8 \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \\ &= 0,5 \end{aligned}$$

Example (3)

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

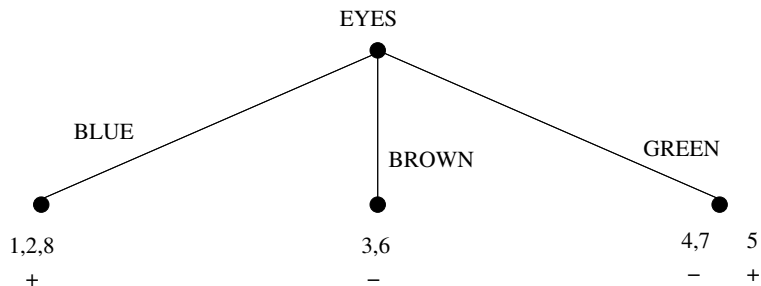
$$G(X, \text{eyes}) = 1 - 0,344 = 0,656 *$$

$$G(X, \text{hair}) = 1 - 1 = 0$$

$$G(X, \text{height}) = 1 - 0,5 = 0,5$$

Example (4)

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



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	-

Example (6)

$$I(X, C) = -1/3 \cdot \log(1/3) - 2/3 \cdot \log(2/3) = 0,918$$

$$\begin{aligned} E(X, \text{hair}) &= (\text{blond}) 1/3 \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \\ &+ (\text{black}) 2/3 \cdot (-1/2 \cdot \log(1/2) - 1/2 \cdot \log(1/2)) \\ &= 0,666 \end{aligned}$$

$$\begin{aligned} E(X, \text{height}) &= (\text{tall}) 1/3 \cdot (0 \log(0) - 1 \cdot \log(1)) \\ &+ (\text{medium}) 1/3 \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \\ &+ (\text{short}) 1/3 \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \\ &= 0 \end{aligned}$$

Example (7)

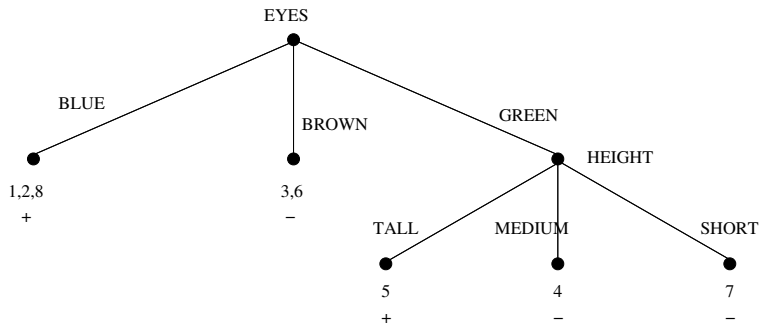
Now the attribute that maximizes the information gain is **eyes**.

$$G(X, \textit{hair}) = 0,918 - 0,666 = 0,252$$

$$G(X, \textit{height}) = 0,918 - 0 = 0,918^*$$

Example (8)

Now the tree can discriminate all the examples



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

Shortcomings of the information gain measure - Example

A ₁	n. exam
a	20+
b	25+
c	20-
d	15-
e	5+ 15-

A ₂	n. ejem
a	40+
b	40-
c	10+ 10-

$$G(A_1) = 1 - \left(\overbrace{\frac{1}{5} \cdot 0}^a + \overbrace{\frac{1}{4} \cdot 0}^b + \overbrace{\frac{1}{5} \cdot 0}^c + \overbrace{\frac{3}{20} \cdot 0}^d + \overbrace{\frac{1}{5} \cdot \left(-\frac{1}{4} \cdot \log \frac{1}{4} \right) - \frac{3}{4} \cdot \log \frac{3}{4}}^e \right) = 0,84$$

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

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:

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

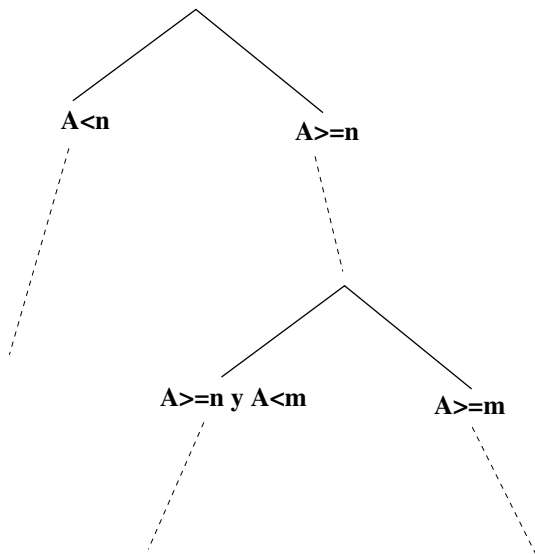
Numerical Attributes

- 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:** To partition the range of values in two or more intervals
- In practice this means to discretize the attribute in two or more values

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

Numerical attributes - Node decisions



Distribution of numerical attributes

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

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)

Missing values - Solutions

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

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

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

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

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 χ^2 with $(c - 1) \cdot (v - 1)$ degrees of freedom

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 not in the majority class and B the binomial probability distribution function, the estimated error for a node can be computed as:

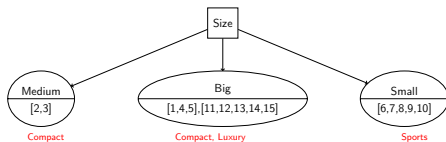
$$Error = N \cdot B_{cf}(E, N)$$

- Where fc 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

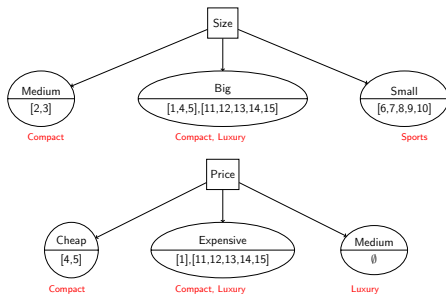
Post-pruning - Example (I)

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

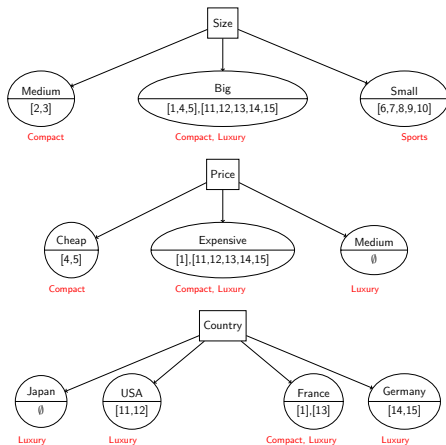
Post-pruning - Example (2)



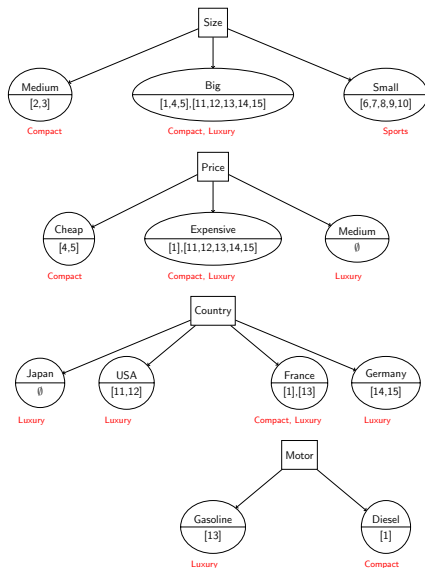
Post-pruning - Example (2)



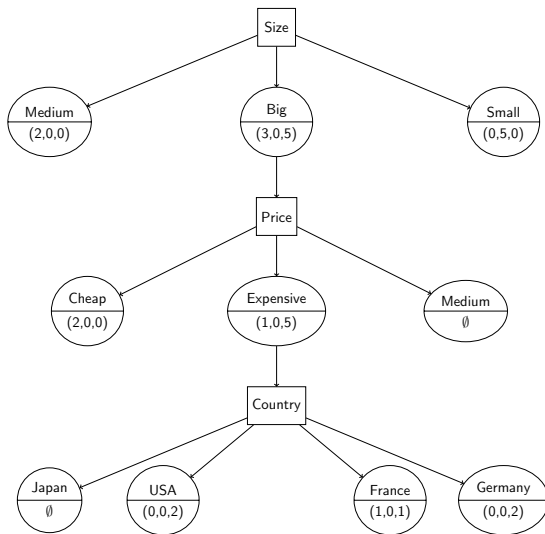
Post-pruning - Example (2)



Post-pruning - Example (2)



Post-pruning - Example (3)



Post-pruning - Example (4)

- Node Expensive

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

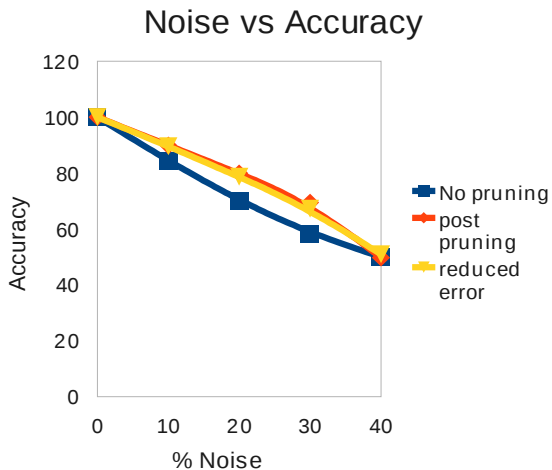
$$Error_{\{USA, Ger, Fr\}} = 2 \cdot B_{0,25}(0, 2) + 2 \cdot B_{0,25}(0, 2) + 2 \cdot B_{0,25}(1, 2) = 4,125$$

- Node Big

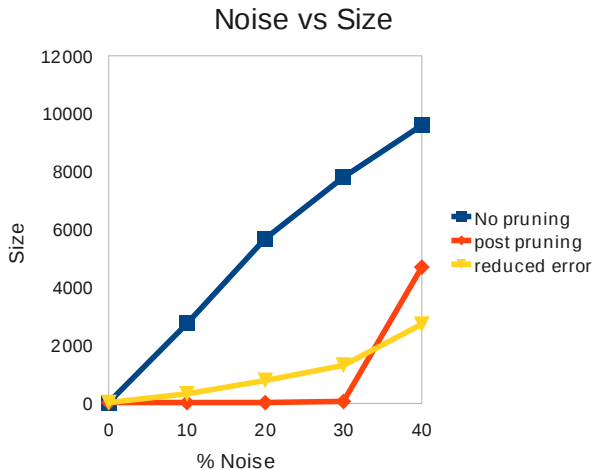
$$Error_{big} = 8 \cdot B_{0,25}(3, 8) = 8 \cdot 0,8862 = 7,089$$

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

Pruning / Noise and % accuracy

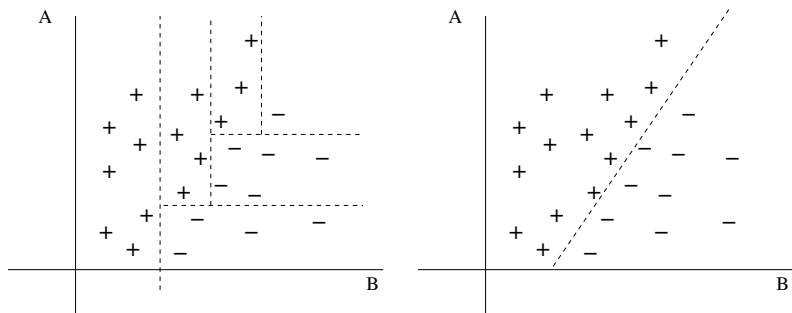


Pruning / Noise and tree size



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



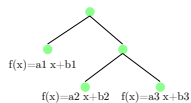
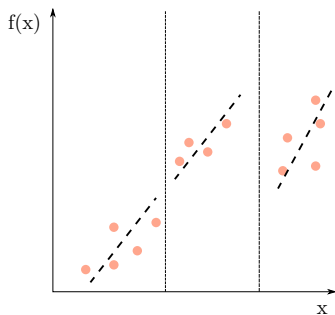
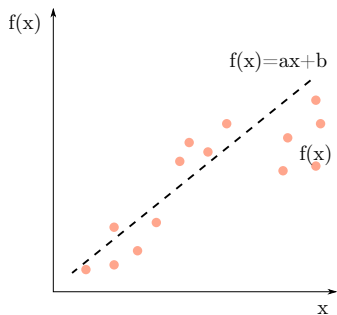
Other algorithms for decision trees

- Different measures for choosing split attributes:
 - CART (Classification And Regression Trees): GINI index
$$G(X, C) = 1 - \sum P(X|C)^2$$
- Different conditions on nodes:
 - Oblique Classifiers: Linear combination of attributes
- Complex predictions schemes on leaves
 - Model trees: Decision trees where the leaves are a model obtained by other ML algorithm
- Prediction of continuous labels
 - Regression trees

Regression trees

- Sometimes we want to predict a function instead of a set of discrete values
- There are several methods for approximating functions given a set of examples
- Usually a specific global model is assumed (eg. linear regression = linear function)
- Decision trees allow to divide the problem in several subproblems, each one can have a different model (or different instances of the same model)

Regression trees



Regression trees

- Splitting function: minimization of the sum of squared error

$$sq_error = \sum_{\forall x} (f(x) - \hat{f}(x))^2$$

- When to stop splitting? \Rightarrow Penalization to the complexity of the model
 - Size of the leaves
 - Function of the size of the model
- Model at the leaves (computational cost!)
 - Mean value
 - Regression model (eg: linear regression)

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 once like in a decision tree
- Some of these algorithm can learn first order rules

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

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

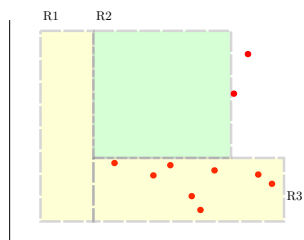
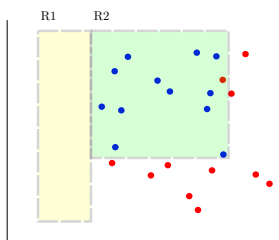
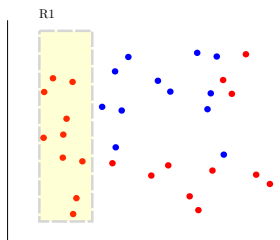
 examples = examples - correct-examples(rule, examples)

 rule = learn-one-rule(attributes,examples)

end

list_rules = order-by-accuracy(list_rules)

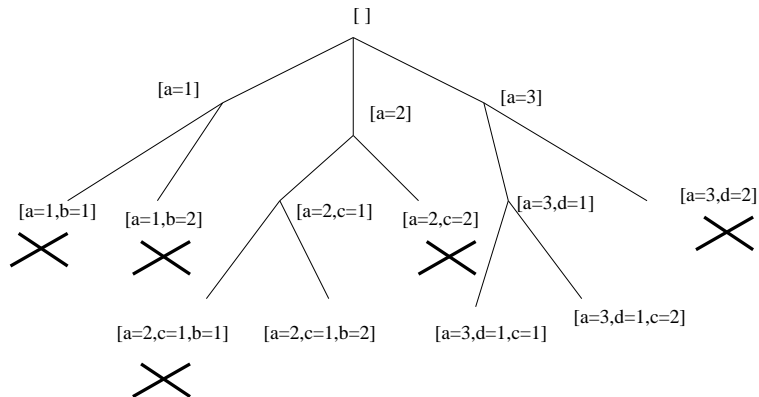
Sequential covering algorithms



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

Example



Other algorithms for rule induction

- The sequential method has its limitations, there are a large variety of alternatives
- For instance we can change how the instances are selected for covering (focusing on the classes)
- There are also other formalisms of rules that can be used
 - Decision tables
 - Decision lists
 - Ripple down rules
 - Default rules

Inductive Logic Programming (ILP)

- Decision trees and rule algorithms can be seen as propositional rule learners
- These algorithms can be extended for learning first order rules (rules with variables)
- These algorithms allow to learn concepts in domains where knowledge can not be expressed as a set of attribute-value pairs
- This extends the complexity of the concepts that can be learned (relational knowledge, sub structures, programs, ...)

Inductive Logic Programming (ILP)

- Imagine the following set of examples:

$number_1$	$number_2$	$number_3$	$larger_{12}$	$larger_{23}$	class
5	4	3	true	true	+
7	4	1	true	true	+
8	5	2	true	true	+
8	9	2	false	true	-
5	3	4	true	false	-
...

- and we want to learn the concept $number_1 >_{trans} number_3$
- an induction tree learner will obtain rules like:

$$number_1 > 5 \wedge number_3 < 3 \wedge number_2 = 4 \rightarrow class = +$$

- when we really want:

$$number_1 > number_2 \wedge number_2 > number_3 \rightarrow number_1 >_{trans} number_3$$

Inductive Logic Programming (ILP)

- We need to represent adequately the examples:
 - number(1), number(2), number(3), ...
 - larger(5,4), larger(4,3),larger(7,4) , ...
 - \neg larger(8,9), \neg larger(3,4), ...
- We need to represent the target predicate:
 - largetrans(5,4,3), largetrans(7,4,1), ...
 - \neg largetrans(8,9,2), \neg largetrans(5,3,4), ...
- We need a bias to reduce the search space (concepts we want to learn)
- Usually the target rules are *Horn clauses*

$$H \vee \neg L_1 \vee \neg L_1 \cdots \vee \neg L_n \iff (L_1 \wedge \cdots \wedge L_n) \rightarrow H$$

FOIL

- FOIL is an algorithm able to learn first order rules
- The idea is similar to the algorithms for rule learning and decision trees
 - 1 We begin with the most general rule (\rightarrow *concept*)
 - 2 Literals are generated and tested to specialize the rule
 - 3 The best candidate is chosen and added to the rule
 - 4 Examples covered by the rule are eliminated

FOIL

- Candidate literals generation
 - Positive or negative literals used in the examples with variables (including the goal), at least one variable from a previous literal or from the head of the rule
 - Equality or inequality literals using the variables from the literals
 - Arithmetic comparisons
- Evaluating the literals/rules
 - Several measures can be used (eg. Information gain, impurity)
 - Length of the rule (to avoid overfitting)

FOIL - example

We have the following examples:

father(john,mary), father(john,peter), father(paul,john),
father(paul,emma), father(harry,matt)
mother(emma,harry), mother(emma,james), mother(mary,jane)
grandfather(paul,peter), grandfather(john,jane), grandfather(paul,mary),
grandfather(paul,harry), grandfather(paul,james)

The goal concept is grandfather

We assume negation by failure (other possible combination of constants and grandfather are false) (9 constants \rightarrow 81 combinations, 5 are positive examples, 76 are negative)

FOIL - example

- We begin with the most general rule $\rightarrow grandfather(x, y)$
- We have several candidate clauses (we only evaluate a subset)
 $\{father(x,y), father(x,z), father(z,y), mother(x,y), mother(x,z), mother(z,y)\}$
 - $father(x, y) \rightarrow grandfather(x, y)$ covers 0 positives (and 5 negatives)
 - $father(x, z) \rightarrow grandfather(x, y)$ covers 5 positives (and 22 negatives)
 - $father(z, y) \rightarrow grandfather(x, y)$ covers 2 positive (and 43 negatives)
 - $mother(x, y) \rightarrow grandfather(x, y)$ covers 0 positives (and 3 negatives)
 - $mother(x, z) \rightarrow grandfather(x, y)$ covers 0 positives (and 18 negatives)
 - $mother(z, y) \rightarrow grandfather(x, y)$ covers 3 positives (24 negatives)

FOIL - example

- We continue with the rule $father(x, z) \rightarrow grandfather(x, y)$
- We have several candidate clauses (we only evaluate a subset)
{ $father(x, y), father(z, y), father(y, z), mother(x, y), mother(z, y), mother(y, z)$ }
 - $father(x, z) \wedge father(x, y) \rightarrow grandfather(x, y)$ covers 0 positives (and 4 negatives)
 - $father(x, z) \wedge father(z, y) \rightarrow grandfather(x, y)$ covers 2 positives (and 0 negatives)
 - $father(x, z) \wedge father(y, z) \rightarrow grandfather(x, y)$ covers 0 positive (and 0 negatives)
 - $father(x, z) \wedge mother(x, y) \rightarrow grandfather(x, y)$ covers 0 positives (and 0 negatives)
 - $father(x, z) \wedge mother(z, y) \rightarrow grandfather(x, y)$ covers 3 positives (and 0 negatives)
 - $father(x, z) \wedge mother(y, z) \rightarrow grandfather(x, y)$ covers 0 positives (and 0 negatives)

FOIL - example

- The best rule is $father(x, z) \wedge mother(z, y) \rightarrow grandfather(x, y)$
- If we delete the positive examples and repeat the process we will obtain the other rule:

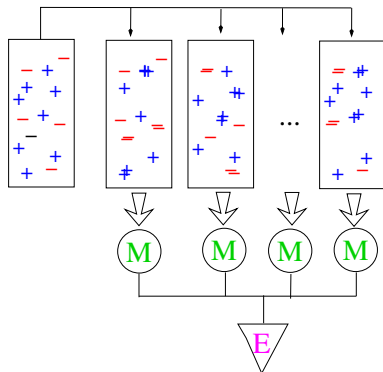
$$father(x, z) \wedge father(z, y) \rightarrow grandfather(x, y)$$

Classifier ensembles/ Meta learning

- An alternative for improving the accuracy of classifiers is to combine the decisions of a group of them (*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*), for example decision trees

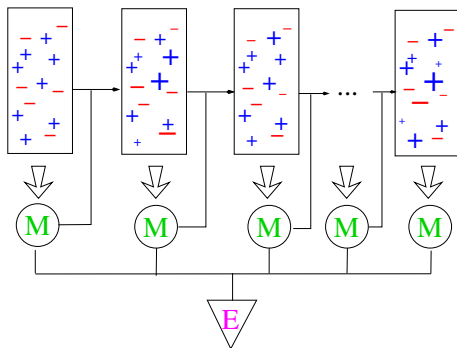
Classifier ensembles - Bagging

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



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.



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.

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.

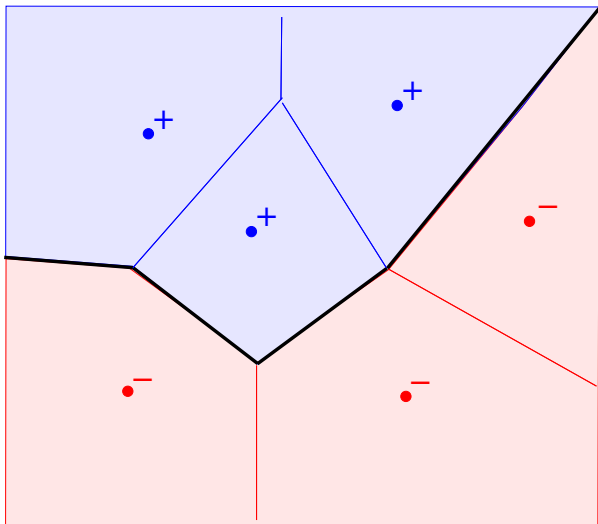
Other hypothesis spaces

- All the algorithms that we have seen assume that the model is some kind of logical expression
- We can use different models in order to learn the concept described by a set of examples
- Changing the model we change our *hypothesis space*
- Different hypothesis space lead to different learning algorithms

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

K-nearest neighbours - hypothesis space (1 neighbour)



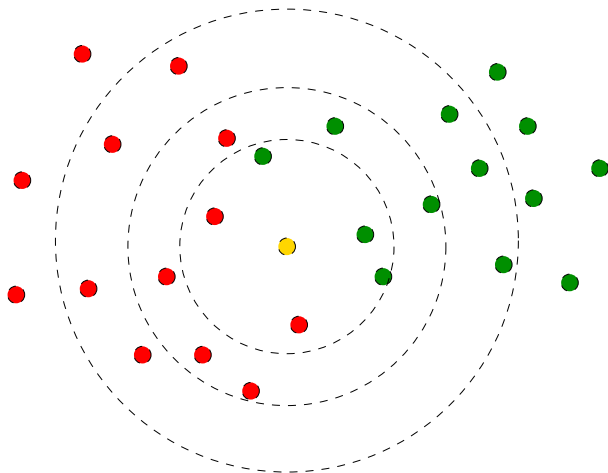
K-nearest neighbours - Algorithm

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

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, ...)
- Once we use weights for the prediction we can relax the constraint of using only k neighbours
 - 1 We can use k examples (local model)
 - 2 We can use all examples (global model)

K-nearest neighbours - Prediction



K-nearest neighbours - Regression

- This learning algorithm can be extended to predict continuous values
- We can just compute the mean of the predictions
- We can fit a model using the k neighbours (Locally Weighted Regression)
 - Minimizing the square error
 - Assuming a specific model: linear function, quadratic function, ...
- We can fit the model locally or globally (using weights depending on the distance)
- Obviously the time complexity will depend on the model we fit and the number of neighbours we use

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

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 and relevance of attributes of the dataset (*curse of dimensionality*) \implies *Attribute Selection*
- The model can not be interpreted (there is no description of the learned concepts)

Bayesian learning

- We suppose that concepts are represented by n-dimensional probability distributions (Hypothesis Space)
- By the Bayes theorem, the probability of a class given an example is:

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

- We will consider as the prediction for a new instance the class with the larger a posteriori probability
- The learning 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
 - The dependence among the variables

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

$$P(C|x) = \frac{P(C) \cdot \prod_{i=1}^n P(x_i|C)}{\prod_{i=1}^n P(x_i)}$$

- $\prod_{i=1}^n P(x_i)$ is independent of the class and can be eliminated
- We estimate separately each $P(x_i|C)$ from the data

Naive Bayes - Prediction

- To predict the class for a new example we only need to find out the class that has the maximum probability
- Given an example $y = (x_1, x_2, \dots, x_n)$, the class for y is:

$$\text{class}(y) = \underset{C}{\operatorname{argmax}} \left[P(C) \cdot \prod_{i=1}^n P(x_i|C) \right]$$

- Surprisingly this approach many times works better than more complex methods

Bayesian learning - Probability estimation (NB)

- For discrete attributes we can estimate $P(x_i|C)$ as the frequency of the value of the attribute in the dataset (multinomial distribution)
- For discrete data with small samples estimation can be problematic (values with zero probability)
 - **LaPlace correction** (n_c frequency of the value in the class, n number of examples, p a priori probability (for instance uniformly distributed), m weighting constant)

$$P(x_i|C) = \frac{n_c + m \cdot p}{n + m}$$

Bayesian learning - Probability estimation (NB)

- For continuous attributes we can estimate $P(x_i|C)$ assuming that they follow a continuous distribution (for instance gaussian) and estimate its parameters from the dataset
- Continuous distributions can be also estimated using locally weighted regression (Kernel Density Estimation)
 - Density in a specific point is a weighted sum of the density of the surroundings

$$P(x_i|C) = \sum_{x_j \in \mathcal{X}} K(d(x_i, x_j))$$

Where $K(d(x_i, x_j))$ is a decreasing function of the distance (usually the gaussian function)

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 represents the attributes