Artificial Intelligence

academic year 2025/2026

Giorgio Fumera, Ambra Demontis

Pattern Recognition and Applications Lab
Department of Electrical and Electronic Engineering
University of Cagliari (Italy)

Machine learning

Decision Trees

Decision Trees

Given a classification problem with d attributes $x_1, \ldots, x_d$ with **discrete** and **finite** domains $\mathcal{X}_1, \ldots, \mathcal{X}_d$ (e.g., Boolean attributes), a **Decision Tree** (DT) is a **tree graph** that represents a **mutually exclusive** set of classification rules of the form:

`IF` *condition* `THEN` *class*

where

- *condition* is a Boolean expression (also called *test*) consisting of the **conjunction** (`AND`) of one or more conditions of the form $x_i = v_i$, where $i \in \{1, \ldots, d\}$ and $v_i \in \mathcal{X}_i$
- *class* is the class label predicted by the DT for any instance which fulfils *condition*

A problem that can be addressed with decision trees

Consider an e-mail spam filtering problem with class labels $\mathcal{Y} = \{\texttt{Spam}, \texttt{Legitimate}\}$ and Boolean attributes denoting the occurrence of the following $d = 5$ terms in an input e-mail:

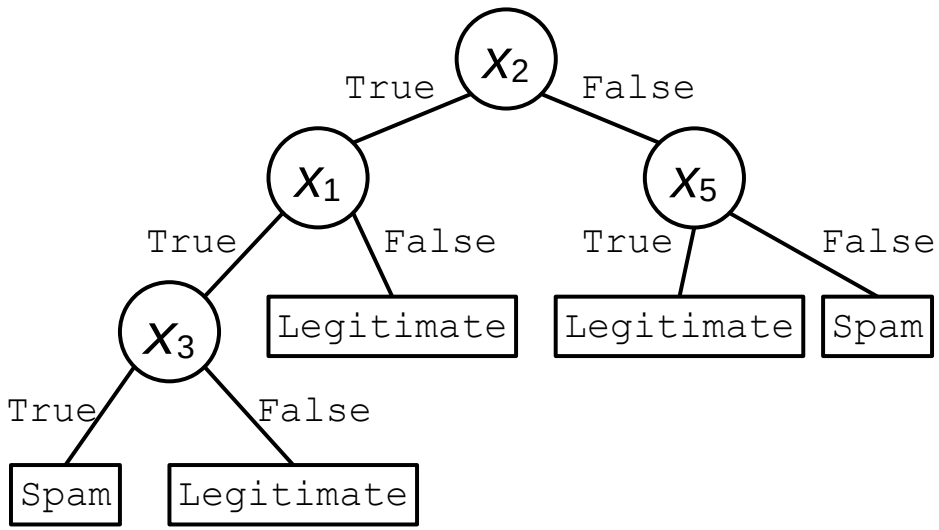
$$A = \{\texttt{cheap}, \texttt{buy}, \texttt{bargain}, \texttt{university}, \texttt{conference}\}$$

For instance, if an e-mail contains the terms `cheap` and `bargain` and no other term in A, its feature vector representation is:

$$\mathbf{x} = (x_1, \ldots, x_5) = (\texttt{True}, \texttt{False}, \texttt{True}, \texttt{False}, \texttt{False})$$

Decision Trees: an example

The following is a possible DT (not necessarily an accurate one):



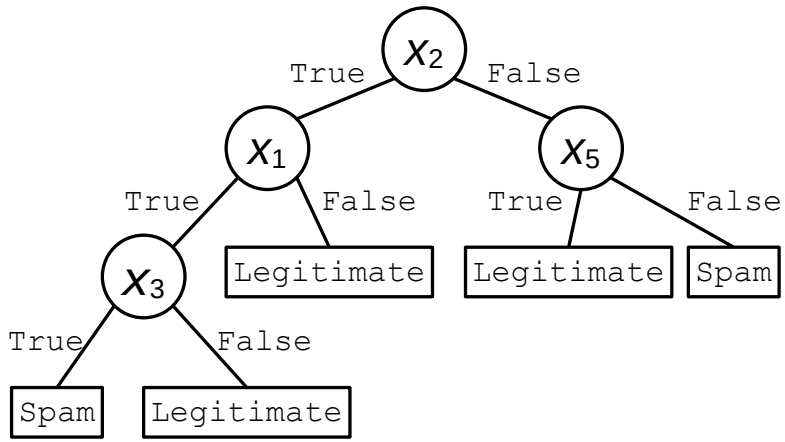
Decision Trees

Formally:

- every **non-leaf node** represents a test on the value of a **single** attribute x_i, and has one successor for each value of its domain $\mathcal{X}_i$
- every **edge** corresponds to **one distinct** value of the attribute in the **parent** node, i.e., to one possible outcome of the corresponding test
- every **leaf node** represents **one** class label (several leaves can share a same class label)
- every **path** from the root node to a leaf represents one **distinct** classification rule
- every attribute can appear **at most once in a path** from the root to a leaf node

An example of path (classification rule)

Non-leaf nodes are represented by circles and leaf nodes by rectangles:



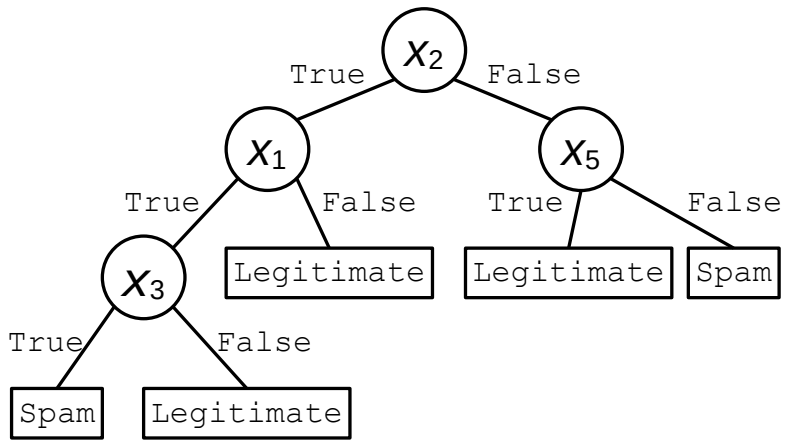
For instance, the classification rule represented by the leftmost leaf node can be rewritten as:

$$\texttt{IF } x_2 = \texttt{True } \texttt{ AND } x_1 = \texttt{True } \texttt{ AND } x_3 = \texttt{True } \texttt{THEN Spam}$$

which can be rephrased as: *if the terms* `buy`, `cheap` and `bargain` *appear in an e-mail, then that e-mail is* `Spam`.

Decision Trees: an example of classification

Given the decision tree and the feature vector representing an email:



$$\mathbf{x} = (x_1, \ldots, x_5) = (\texttt{True}, \texttt{False}, \texttt{True}, \texttt{False}, \texttt{False})$$

To classify a sample we follow the path corresponding to the sample feature values.

Decision Trees

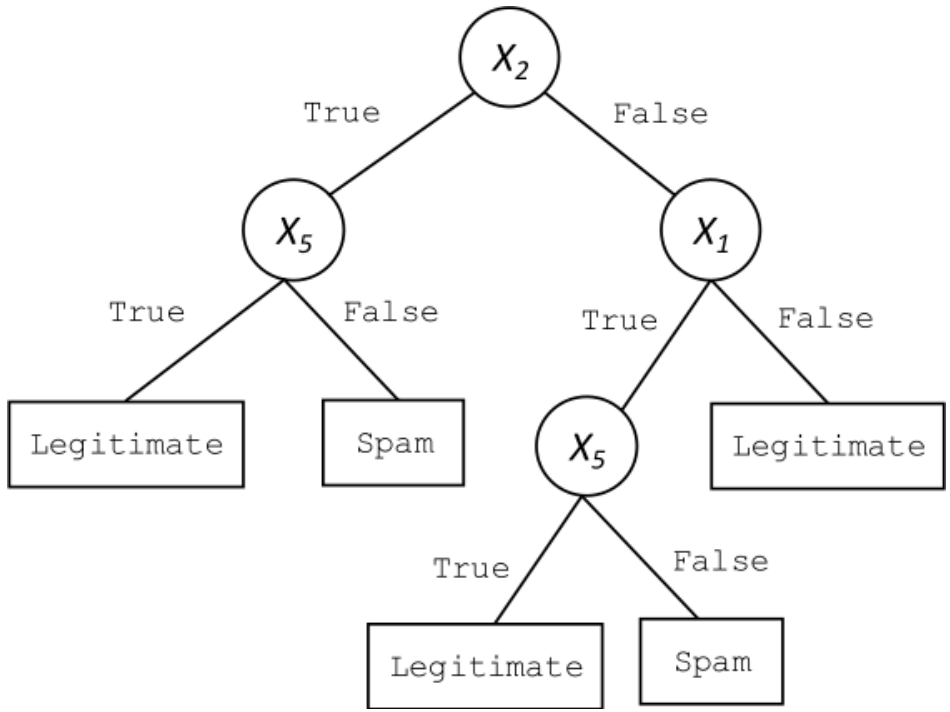
The above properties imply that the classification rules represented by a DT are **complete** and **mutually exclusive**, i.e., exactly one of them applies to any given feature vector.

Note also that some attributes may not appear in a DT, i.e., the corresponding classification rules may not take into account some attributes.

On the other hand, every attribute may appear in **different** paths, i.e., in different classification rules (as a particular case, the attribute associated to the root node appears in **all** such rules).

Decision Trees: another an example

An example of DT with an attribute (x_5) that appears in two different paths.



Note also that x_3 and x_4 do not appear in the above DT.

Toward a learning algorithm for Decision Trees

If DTs are used for a given supervised classification problem, the **hypothesis space** (**classifier model**) $\mathcal{H}$ is the set of all possible, distinct DTs that can be constructed using the class labels $\mathcal{Y}$ and the attribute space $\mathcal{X}$.

The goal of a DT learning algorithm is therefore to select a specific DT $h \in \mathcal{H}$, based on a training set $\mathcal{T}$.

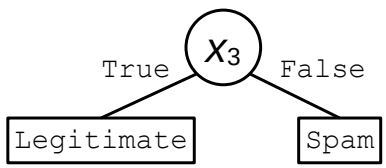
According to the general principles of inductive inference, this amounts to select the **"simplest"** DT **consistent** with $\mathcal{T}$.

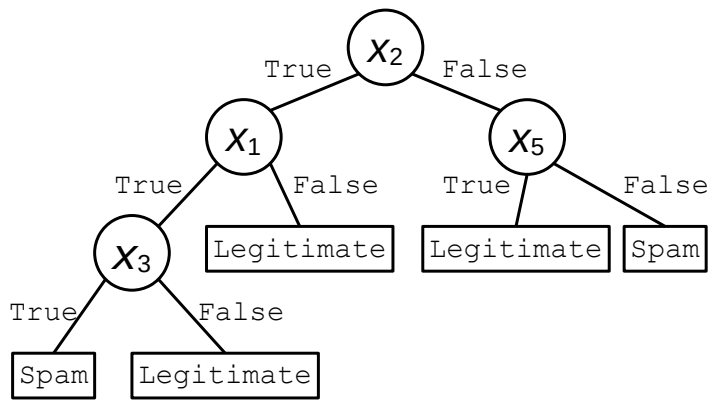
To rigorously define the above goal, it is first necessary to define a criterion to evaluate the **"complexity"** of a DT.

Toward a learning algorithm for Decision Trees

A possible, intuitive measure of the "complexity" of a DT is the number of its **non-leaf** (internal) **nodes**.

Accordingly, the simplest DTs are the ones made up of a root node whose successors are all leaf nodes (called **decision stumps**), as in the example below on the left, which refers to the previous spam filtering problem.





The complexity of decision stumps is equal to 1. The DT above on the right has instead a complexity of 4.

Toward a learning algorithm for Decision Trees

How to devise a **learning algorithm** that finds the smallest DT consistent with the training set at hand?

A seemingly straightforward solution, is to first build **all** DTs of size 1, then **all** DTs of size 2, and so on, until a consistent DT is found.

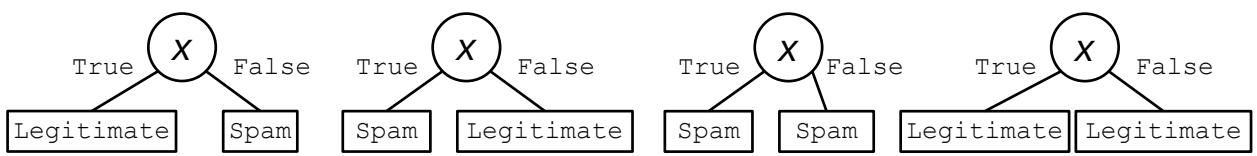
However, what is the **computational complexity** of the above **brute-force** solution? In other words, **how many** different DT sizes exist, and **how many** DTs of size 1, 2, etc. can be built, for a given classification problem?

Toward a learning algorithm for Decision Trees

To have an idea of the computational complexity of the learning algorithm sketched above, consider the simplest case of a classification problem with d binary (e.g., Boolean) attributes and $m = 2$ classes, e.g., the e-mail spam filtering problem in the previous example.

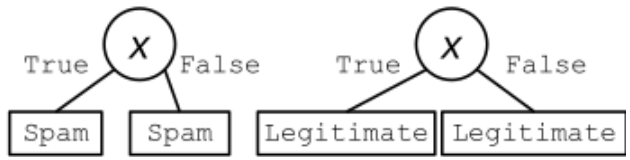
First, how many distinct DTs of **size 1** (decision stumps) can be built?

- the root node can be associated with d **different attributes**, and always has exactly two successors (leaf nodes), with edges labelled `True` and `False`
- for each attribute x at the root node, the number of **disposition with repetition** of the class labels in the corresponding leaf nodes is equal to 4:



This amounts to $4d$ distinct DTs of **size 1**.

Please note that having a not leaf node that leads to the following configurations would not be useful at all:

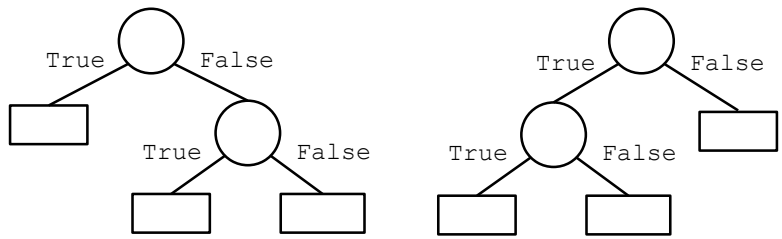


However, they are possible DTs, therefore we should count them too.

Toward a learning algorithm for Decision Trees

What about the number of distinct DTs of **size 2**?

- two possible DT structures can be obtained



- for each structure, there are $d(d-1)$ choices for the attributes of the two nodes (by definition each attribute can appear at most once in a same path from the root to a leaf)...
- ...and 2^3 way to dispose the 2 class labels in the 3 leaf nodes

This amounts to $2 \times d(d-1) \times 2^3 = 2^4 d(d-1)$ distinct DTs of **size 2**.

Toward a learning algorithm for Decision Trees

- What about the number of distinct DTs of size 3, 4, etc.?
- And, what about the **largest** DTs?
  - all their leaf nodes are at depth d, since each attribute can appear at most once in a same path from the root to a leaf.
  - *the number of not-leaf nodes?*

Toward a learning algorithm for Decision Trees

- The largest DT will be *perfect* binary decision trees (a tree with 2 edges for each not-leaf node and all the labels at the same level).
- The number of nodes from the root to depth n (equal to $d-1$, which is the maximum depth at which we have not-leaf node) is equal to the summation of the first **n-1** terms of a finite geometric series: $S_n = a\ r^0 + a\ r^1 + a\ r^2 + ... + a\ r^n = \sum_{k=0}^{n} a\ r^k$
- Therefore, it can be computed in closed form.
- If $r = 1$, then $S_n = a\ (n+1)$
- Otherwise $S_n = a\ (\frac{1-r^{n+1}}{1-r})$
- In our case $a = 1$ and $r = 2$, thus, $S_{n-1} = 2^n - 1$

Toward a learning algorithm for Decision Trees

- What about the number of distinct DTs of size 3, 4, etc.?
- And, what about the **largest** DTs?
  - all their leaf nodes are at depth d, since each attribute can appear at most once in a same path from the root to a leaf.
  - *the number of not-leaf nodes is* $1 + 2 + 4 + \ldots + 2^{d-1} = 2^d - 1$
  - *the number of leaf nodes is* 2^d
  - to compute the number of possible DTs, one should consider the possible distinct dispositions of attributes at inner nodes and of class labels at the leaf nodes
  - already a number of leaf nodes $l = 20$ would lead to $2^l = 1048576$ possible dispositions of labels ...
- Finally, what about problems with $m > 2$ classes (e.g., in handwritten digit recognition, $m = 10...$) and attributes whose domains are made up of more than two values?

Toward a learning algorithm for Decision Trees

The above examples should make it clear that the **brute-force** learning algorithm sketched above has a **huge computational complexity**.

More formally, it has been shown that finding the smallest consistent DT (e.g., by constructing **all** DTs of size 1, 2, etc., until a consistent one is found) is a **NP-hard** problem.

A naive learning algorithm for Decision Trees

One may wonder whether focusing just on finding a **consistent** DT, **disregarding its size** (i.e., the **minimal complexity** requirement), may allow to devise a DT learning algorithm characterised by a low computational complexity.

It turns out that a very simple solution exists to this problem:

1. for each training example $(\mathbf{x}, y)$:
   - 1.1 build a **distinct** path of depth d from the root to a leaf node, including all the d attributes with their values $\mathbf{x}_1, \ldots, \mathbf{x}_d$, in **any** order (consistently with previously built paths, if any)
   - 1.2 label the leaf node with y
2. add a leaf note to each **incomplete** branch (if any), and associate it a **randomly** chosen class label in $\mathcal{Y}$

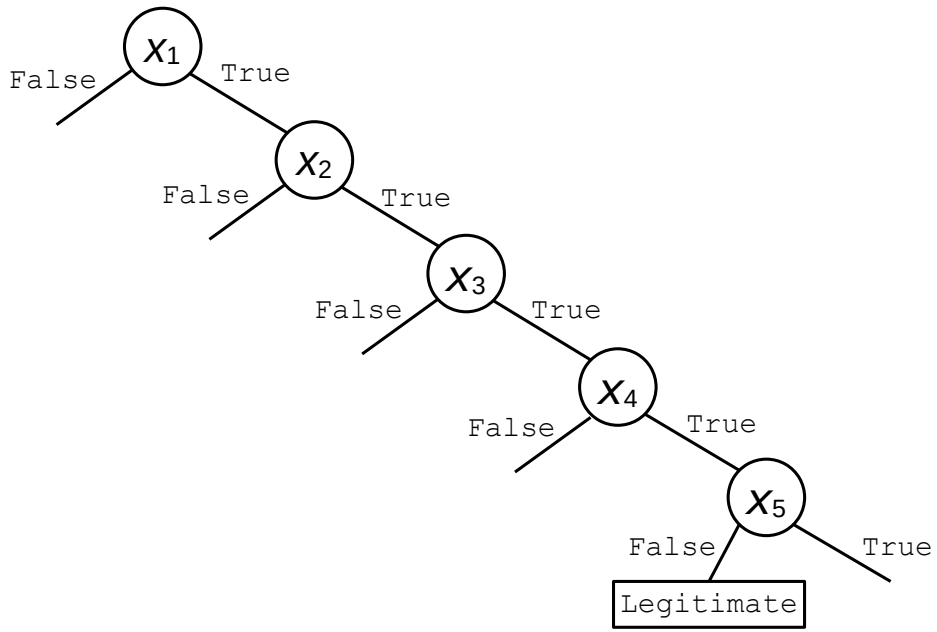
A naive learning algorithm for Decision Trees

As an example, consider again the above Boolean attributes for an e-mail spam filtering problem, associated to the terms `research`, `cheap`, `travel`, `conference` and `price`, and assume that a training set $\mathcal{T}$ of ten e-mails, six legitimate (L) and four spam (S), has been collected for building a DT, with the following attribute values:

ID	y	x_1	x_2	x_3	x_4	x_5
m_1	L	T	T	T	T	F
m_2	L	T	F	T	F	T
m_3	L	T	T	F	T	F
m_4	L	F	F	F	T	T
m_5	L	F	F	T	F	F
m_6	L	T	T	T	F	T
m_7	S	F	T	T	T	F
m_8	S	F	T	F	T	F
m_9	S	T	F	F	F	T
m_{10}	S	F	T	T	T	T

A naive learning algorithm for Decision Trees

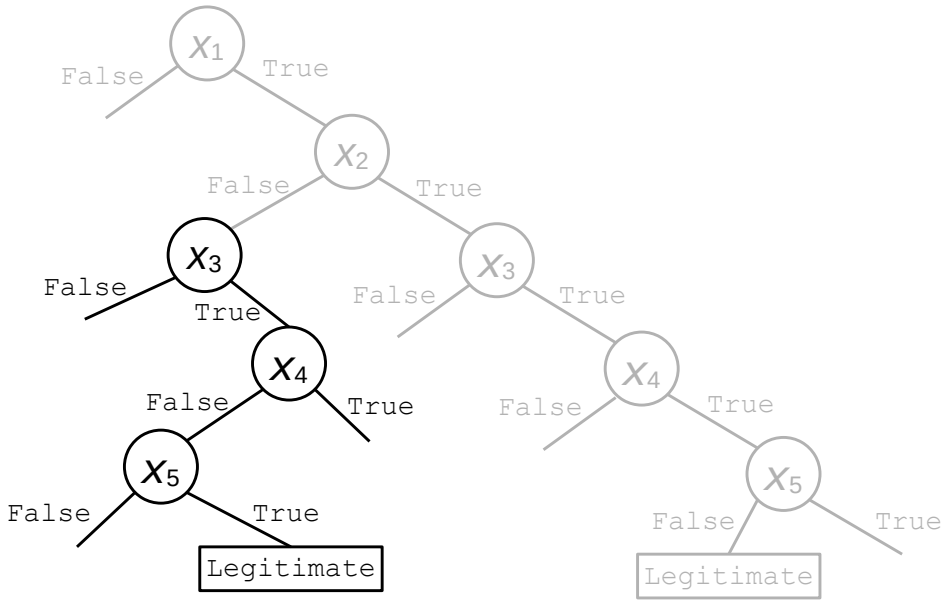
If the above algorithm is applied starting from $\mathtt{m}_1$ (a legitimate e-mail), and processing the attributes in the order x_1, x_2, x_3, x_4, x_5, the following **partial** DT is obtained:



Note that, regardless of how the other branches will be completed, this DT is guaranteed to correctly classify $\mathtt{m}_1$.

A naive learning algorithm for Decision Trees

If the next example processed by this algorithm is $\mathtt{m}_2$ (another legitimate e-mail), the previous branch $x_1 = \mathtt{True}$ can be initially exploited (since also for $\mathtt{m}_2$ the value of x_1 is $\mathtt{True}$), then it is necessary to build a new branch starting from $x_2 = \mathtt{False}$:



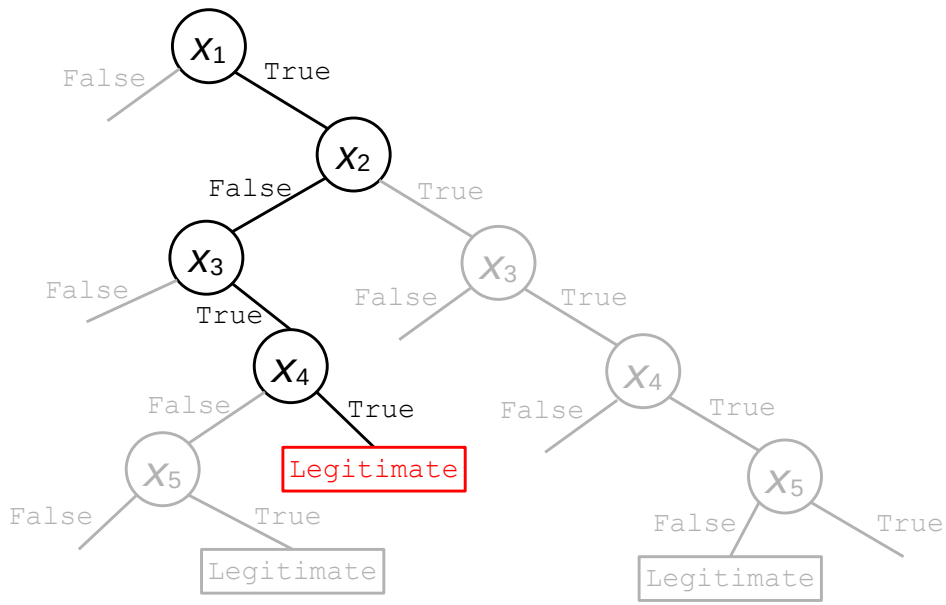
The updated, partial DT is guaranteed to correctly classify both $\mathtt{m}_1$ and $\mathtt{m}_2$.

A naive learning algorithm for Decision Trees

The learning algorithm proceeds similarly on each of the remaining training examples.

After that, some branches remain incomplete, e.g., the branch corresponding to $x_1 = \texttt{True}, x_2 = \texttt{False}, x_3 = \texttt{True}, x_4 = \texttt{True}$, since no training example exhibits the corresponding attribute values.

A leaf node with a randomly chosen class label is therefore added to complete such branches, as in the example on the right (for simplicity, the complete branches corresponding to each training example are not shown):

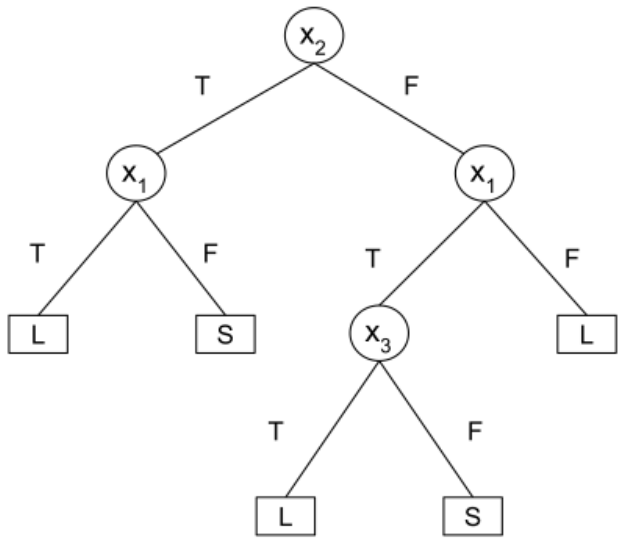


A naive learning algorithm for Decision Trees

The above learning algorithm guarantees consistency with $\mathcal{T}$ (unless there are examples of different classes with identical attribute values), but it disregards DT size.

What about its generalisation capability?

Note first that DTs consistent with $\mathcal{T}$ and **much smaller** than the ones produced by the above learning algorithm usually exist. In the above toy example it is easy to **manually** build some of them, like the following one:



Which DT is more likely to generalise to unseen e-mails?

A naive learning algorithm for Decision Trees

Note again that the above learning algorithm builds a **specific** DT branch, i.e., a **specific** `IF...THEN...` classification rule, for each training example. In practice, the resulting DT just **memorises** the training examples.

Since each classification rule is "supported" by a **single** example, the resulting DTs:

- are **very unlikely** to capture the main distinctive characteristics of the different classes: instead, they **over-fit** the training set, and exhibit a **poor** generalisation capability to unseen examples
- tend to be **much larger** than necessary: much smaller, but still consistent DTs exist

This is a concrete example, specific to DTs, that highlights the importance of the **minimal complexity** principle to prevent over-fitting.

Effective learning algorithms for Decision Trees: ID3

A **trade-off** between computational complexity and generalisation capability is therefore required toward devising a **"good enough"** DT learning algorithm.

A common solution is to construct a **"reasonably small"**, consistent DT, using some low-complexity **heuristics** that favour smaller DTs over larger ones.

This approach is obviously **suboptimal**, since it does not guarantee to find the **smallest** consistent DT; nevertheless, it allowed to devise both **effective** and **efficient** DT learning algorithms.

In the following, one of the first and simpler DT learning algorithms will be presented, named **ID3** (Iterative Dichotomizer v. 3), by J.R. Quinlan (1986).

The ID3 learning algorithm

ID3 is a recursive, **top-down** algorithm, that builds a DT from the root to the leaves.

The key intuition is that, to obtain a small and consistent DT, when an edge is reached by training examples of different classes, the inner node to be added should be associated with the **most "discriminant"** attribute for those examples, *i.e., an attribute which splits them as much as possible into subsets of examples of a **same** class*.

Indeed, if all the training examples reaching an edge belong to a same class y, that edge can be immediately completed with a leaf node associated to y.

Sketch of the ID3 learning algorithm

To show how ID3 works, consider first the same training set $\mathcal{T}$ of the previous example, with an additional Boolean attribute x_6 corresponding, e.g., to the term `algorithm`:

ID	y	x_1	x_2	x_3	x_4	x_5	x_6
m_1	L	T	T	T	T	F	T
m_2	L	T	F	T	F	T	T
m_3	L	T	T	F	T	F	T
m_4	L	F	F	F	T	T	T
m_5	L	F	F	T	F	F	T
m_6	L	T	T	T	F	T	T
m_7	S	F	T	T	T	F	F
m_8	S	F	T	F	T	F	F
m_9	S	T	F	F	F	T	F
m_{10}	S	F	T	T	T	T	F

Sketch of the ID3 learning algorithm

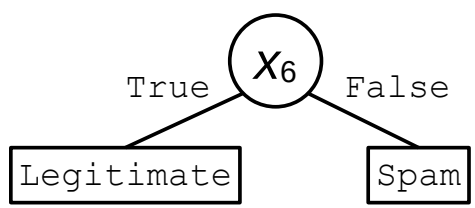
First, ID3 builds the root node: which attribute should be associated to it? Which attribute is most "discriminant" for $\mathcal{T}$?

Note that **all** training examples for which $x_6 = \texttt{True}$ (i.e., $\texttt{m}_1$ to $\texttt{m}_6$), are legitimate, and **all** the ones for which $x_6 = \texttt{False}$ ($\texttt{m}_7$ to $\texttt{m}_{10}$), are spam.

In other words, x_6 in the root node splits $\mathcal{T}$ into subsets whose elements belong to a **single** class, and has therefore the highest possible discriminant capability.

This allows to build a consistent DT of the smallest possible size, i.e., a decision stump (1 inner node), corresponding to two classification rules:

$\texttt{IF } x_6 = \texttt{True THEN Legitimate}$
$\texttt{IF } x_6 = \texttt{False THEN Spam}$



Sketch of the ID3 learning algorithm

However, in real-world problems it is unlikely that a single attribute allows to perfectly split the whole training set.

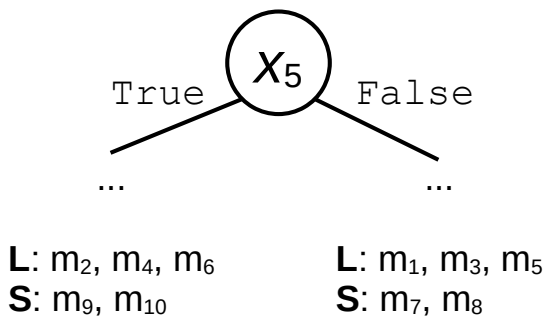
The discriminant capability of all the attributes has therefore to be evaluated and compared.

NB: In the following, only a **qualitative** evaluation will be made. A formal, **quantitative** criterion will be presented later.

Sketch of the ID3 learning algorithm

Consider again the original training set presented above, without x_6.

Let us first evaluate the choice of x_5 for the root node: this corresponds to building two **partial** rules: `IF` $x_5 =$ `True...`, and `IF` $x_5 =$ `False...`, and splits the training examples as shown below:

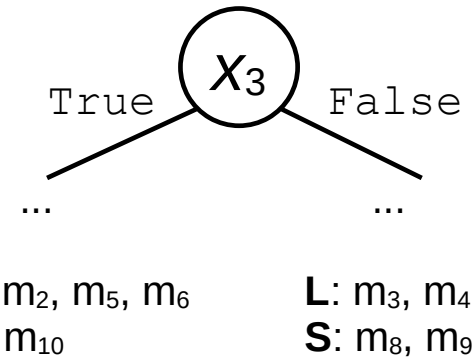


Remember that $\mathcal{T}$ contains 6 legitimate and 4 spam e-mails.
If x_5 is chosen for the root node, it splits $\mathcal{T}$ into two subsets, each one containing 3 legitimate and 2 spam e-mails.

Accordingly, x_5 does not seem a good choice: it does not split $\mathcal{T}$ into subsets with a high predominance of any of the classes.

Sketch of the ID3 learning algorithm

Consider now using x_3 for the root node:



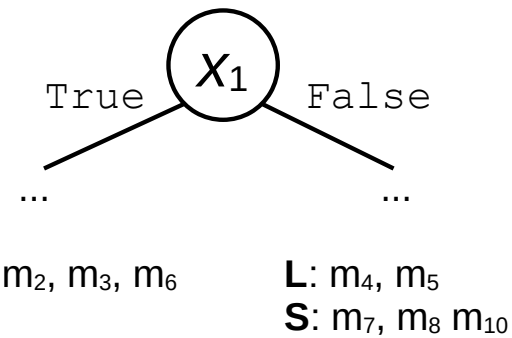
The corresponding split is partly better than x_5 and partly worse:

- for $x_3 = \texttt{True}$, the class ratio is 1:2, which is more skewed toward one of the classes than in the whole $\mathcal{T}$
- for $x_3 = \texttt{False}$, the class ratio is 1:1, with no predominance of any class

Accordingly, x_3 does not look better than x_5.

Sketch of the ID3 learning algorithm

If x_1 is used, instead, the following split is obtained:



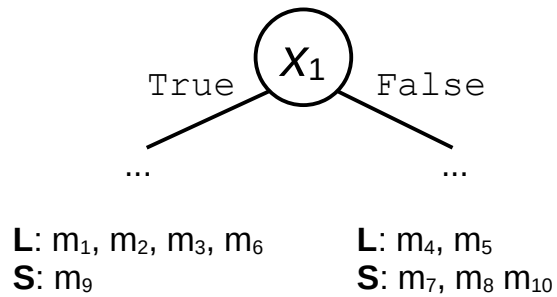
This split is qualitatively better than the previous ones:

- for $x_1 =$ `True`, the class ratio is 1:4
- for $x_1 =$ `False`, the class ratio is 2:3

Sketch of the ID3 learning algorithm

It is not difficult to see that x_2 and x_4 produce splits with identical class ratios, which are not as good as x_1 (this is left as an exercise).

Accordingly, ID3 chooses the attribute x_1 for the root node:



Since both subsets produced by this split contain training examples of **different** classes, it is not possible to get a DT consistent with $\mathcal{T}$ by adding leaf nodes.

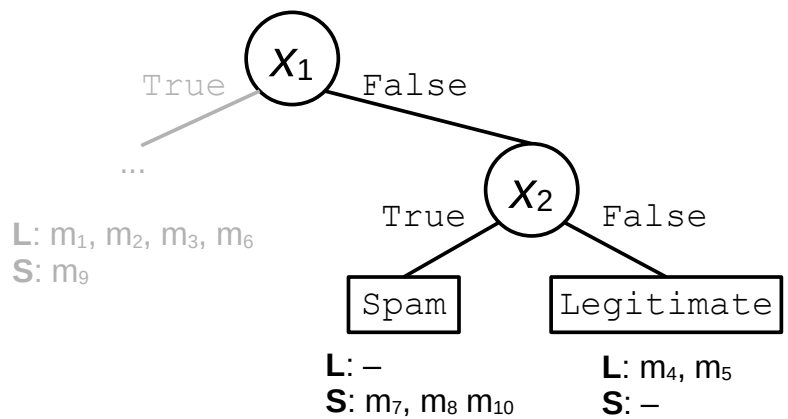
Therefore, ID3 proceeds by **recursively** building a **sub-tree** for both edges, aimed at correctly classifying the corresponding **subsets** of $\mathcal{T}$, using all the attributes **except for** x_1.

Sketch of the ID3 learning algorithm

The sub-tree for the branch $x_1 = \texttt{False}$ is built in the same way, starting from its root; in particular:

- only the training examples that reach this branch are considered (see above): $\texttt{m}_4$, $\texttt{m}_5$, $\texttt{m}_7$, $\texttt{m}_8$ and $\texttt{m}_{10}$
- all the attributes except for x_1 (already used in this branch) are considered

It is easy to see that attribute x_2 perfectly splits **these five examples**, and therefore will be chosen by ID3, which allows to complete both its branches with a leaf node:



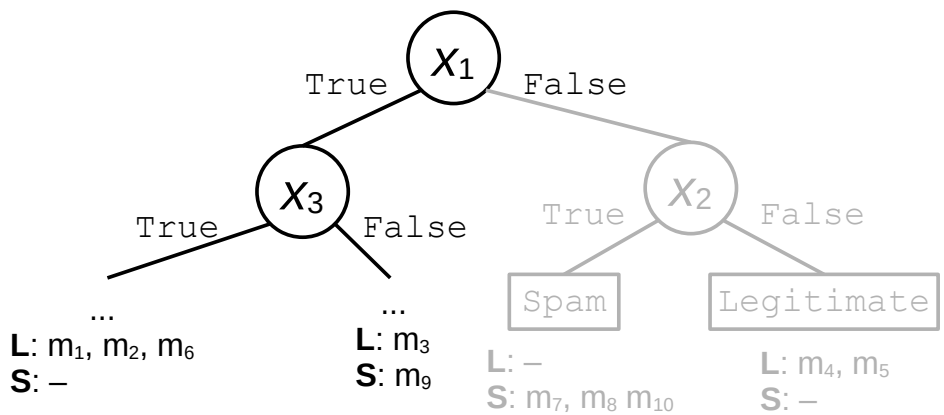
Sketch of the ID3 learning algorithm

ID3 then proceeds on the branch $x_1 = \texttt{True}$ of the root node, focusing this time on $\texttt{m}_1$, $\texttt{m}_2$, $\texttt{m}_3$, $\texttt{m}_6$ and $\texttt{m}_9$.

Although none of the remaining attributes can perfectly split these examples, x_3 looks as a good choice, since the condition

$$x_1 = \texttt{True} \text{ and } x_3 = \texttt{True}$$

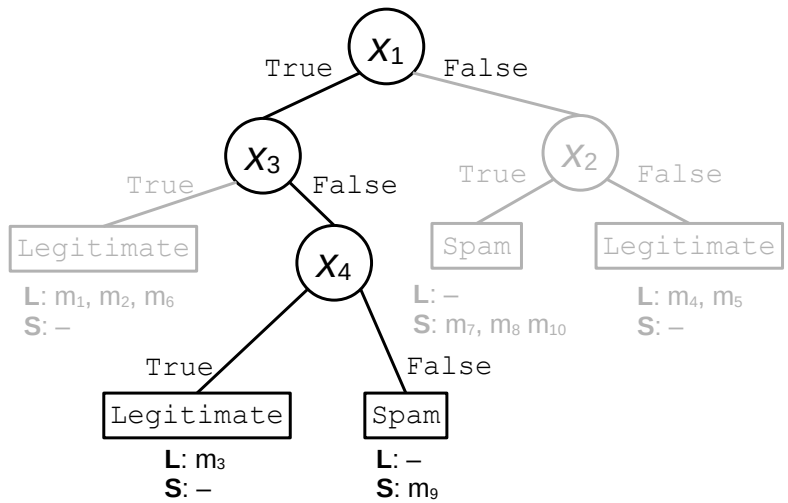
is fulfilled by only legitimate examples:



Sketch of the ID3 learning algorithm

ID3 now proceeds along the branch $x_1 = \texttt{True}$ and $x_3 = \texttt{False}$, which corresponds to $\texttt{m}_3$ and $\texttt{m}_9$, and evaluates the discriminant capability of the remaining attributes, x_2, x_4 and x_5.

Among them, x_4 is chosen, since it perfectly splits $\texttt{m}_3$ and $\texttt{m}_9$:



The DT is now complete, therefore ID3 stops and returns it.

This qualitative example shows that ID3 allows to build much smaller, consistent DTs than the naive learning algorithm previously presented, although it does not guarantee to find the **smallest** DT.

Sketch of the ID3 learning algorithm

A pseudo-code description where only the criterion for evaluating the most discriminant attribute is not described:

function DT_LEARNING ($\mathcal{T}$, A) **returns** a decision tree
    `if` $\mathcal{T}$ is empty `or` A is empty `then`
        build a leaf node L and associate it with *majoritylabel*($\mathcal{T}$)
        `return` L
    `if` $\mathcal{T}$ contains examples of a single class y `then`
        build a leaf node L and associate it with the label y
        `return` L
    root $\leftarrow$ create the root node of a new decision tree
    $x \leftarrow$ the **most discriminant** attribute in A
    associate *root* with x
    $A' \leftarrow A - \{x\}$
    `for each` value $v \in \mathcal{X}$ `do`
        $\mathcal{T}' \leftarrow$ the subset of $\mathcal{T}$ where $x = v$
        subtree $\leftarrow$ DT_LEARNING ($\mathcal{T}'$, A')
        add to *root* an edge labelled as v
        connect appropriately that edge with the root node of *subtree*
    `return` *root*

Sketch of the ID3 learning algorithm

Where *majoritylabel* is a function that get a dataset and, if the dataset is not empty and there is a majority label return that label, otherwhise (if there isn't a majority class or if the dataset is empty), return a random label.

Evaluating attributes' discriminant capability

The heuristic used by ID3 is to associate every inner node `n` of a DT (including the root) with the attribute (among the ones not already used in the same branch) that splits the training examples arriving at `n` into subsets made up of examples belonging "as much as possible" to a single class.

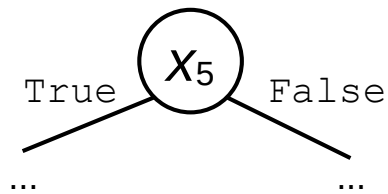
It is therefore necessary to define a measure to evaluate the extent to which the training examples arriving at `n` are split by a given attribute into homogeneous subsets.

To this aim, ID3 uses the **entropy** of a probability distribution function.

Evaluating attributes' discriminant capability

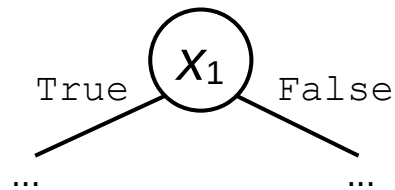
Consider first the root node of a DT, for which the whole training set and all the attributes have to be considered.

For instance, these are two possible choices for the root node, for the training set $\mathcal{T}$ of the previous example:



L: m_2, m_4, m_6
S: m_9, m_{10}

L: m_1, m_3, m_5
S: m_7, m_8



L: m_1, m_2, m_3, m_6
S: m_9

L: m_4, m_5
S: m_7, m_8 m_{10}

The corresponding splitting of $\mathcal{T}$ produces the following class ratios (legitimate : spam)

- x_5: 3 : 2 for both its values
- x_1: 4 : 1 for $x_1 =$ `True`, 2 : 3 for $x_1 =$ `False`

Evaluating attributes' discriminant capability

For a given attribute x, we can compute the **conditional probability density functions** $P(Y|X)$, where Y and X denote **random variables** corresponding to the class label and the value of the attribute x of any **unknown** (random) example.

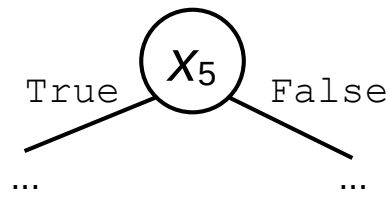
In the spam filtering example, each attribute x splits $\mathcal{T}$ into two subsets corresponding to $x = \texttt{True}$ and $x = \texttt{False}$: therefore $P(Y|X)$, with $Y \in \mathcal{Y} = \{\texttt{Legitimate}, \texttt{Spam}\}$ denotes the **two** conditional probability density functions:

- $P(Y|X = \texttt{True})$
- $P(Y|X = \texttt{False})$

i.e., the probability that an **unknown** (random) e-mail is legitimate (or spam), given that it contains (or does not contain) the term associated to x.

Evaluating attributes' discriminant capability

For instance, consider again the attribute x_5:



L: m_2, m_4, m_6
S: m_9, m_{10}

L: m_1, m_3, m_5
S: m_7, m_8

The estimate of $P(Y|X_5 = \texttt{True})$ is

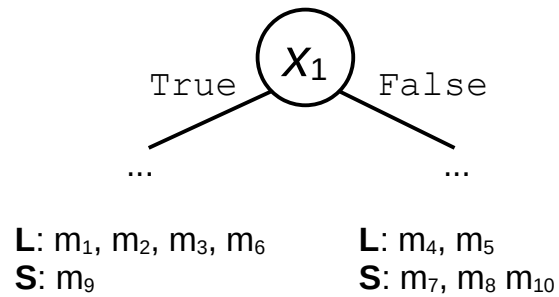
- $P(Y = \texttt{Legitimate}|X_5 = \texttt{True}) = \frac{3}{5} = 0.6$
- $P(Y = \texttt{Spam}|X_5 = \texttt{True}) = \frac{2}{5} = 0.4$

and the estimate of $P(Y|X_5 = \texttt{False})$ is

- $P(Y = \texttt{Legitimate}|X_5 = \texttt{False}) = \frac{3}{5} = 0.6$
- $P(Y = \texttt{Spam}|X_5 = \texttt{False}) = \frac{2}{5} = 0.4$

Evaluating attributes' discriminant capability

Similarly, for the attribute x_1:



The estimate of $P(Y|X_1 = \texttt{True})$ is

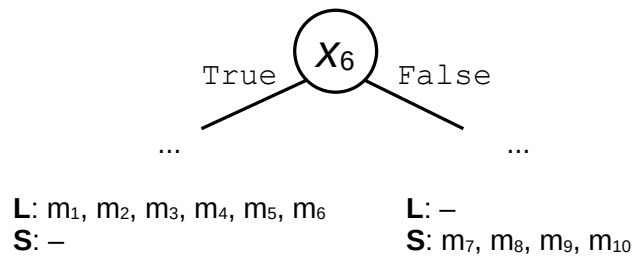
- $P(Y = \texttt{Legitimate}|X_1 = \texttt{True}) = \frac{4}{5} = 0.8$
- $P(Y = \texttt{Spam}|X_1 = \texttt{True}) = \frac{1}{5} = 0.2$

and the estimate of $P(Y|X_1 = \texttt{False})$ is

- $P(Y = \texttt{Legitimate}|X_1 = \texttt{False}) = \frac{2}{5} = 0.4$
- $P(Y = \texttt{Spam}|X_1 = \texttt{False}) = \frac{3}{5} = 0.6$

Evaluating attributes' discriminant capability

As a final example, if also the attribute x_6 considered previously was used, one would obtain a **perfect** split:



The estimate of $P(Y|X_6 = \texttt{True})$ is

- $P(Y = \texttt{Legitimate}|X_6 = \texttt{True}) = \frac{6}{6} = 1$
- $P(Y = \texttt{Spam}|X_6 = \texttt{True}) = \frac{0}{6} = 0$

and the estimate of $P(Y|X_6 = \texttt{False})$ is

- $P(Y = \texttt{Legitimate}|X_6 = \texttt{False}) = \frac{0}{0} = 0$
- $P(Y = \texttt{Spam}|X_6 = \texttt{False}) = \frac{4}{4} = 1$

Evaluating attributes' discriminant capability

If an attribute x has the **highest discriminant capability** (i.e., it **perfectly splits** the training examples), then, for each of its values v

- $P(Y = y_v | X = v) = 1$, for one of the classes y_v
- $P(Y = y | X = v) = 0$, for every other class $y \neq y_v$

Instead, if an attribute x has the **worst discriminant capability**, i.e., it splits the training examples **uniformly** across **all** classes for each of its values v, then:

$$P(Y = y | X = v) = \frac{1}{C} \quad \text{for each } y \ ,$$

where C is the number of classes.

We would like to have a single value that allows us to understand which attribute is more discriminant.

Evaluating attributes' discriminant capability

Remember now that the **entropy** of random variable Y is a measure of **uncertainty** that is associated with its distribution. For a **discrete** random variable Y with **finite** domain $\mathcal{Y} = \{y_1, \ldots, y_C\}$ it is defined as:

$$H(Y) = -\sum_{i=1}^{C} P(Y = y_i) \log_2 P(Y = y_i) \text{ bits.}$$

For example, a fair coin has the same probability to come up heads or tails when flipped. Therefore, its entropy is:

$$H(fair) = -\left(0.5 log_2 0.5 + 0.5 log_2 0.5\right) = 1 \; bit.$$

If instead we consider an unfair coin that comes up head 99% of the time, intuitively, has less uncertainty (if we guess heads we'll be wrong only the 1% of the times):

$$H(unfair) = -\left(0.99 log_2 0.99 + 0.01 log_2 0.01\right) = 0.08 \; bits.$$

Evaluating attributes' discriminant capability

It is known that $H(Y) \in [0, \log_2 C]$:

- if all values y_i are **equiprobable**, i.e., $P(Y = y_i) = \frac{1}{C}$ for all y_i (**maximum uncertainty**), then $H(Y) = \log_2 C$
- if only **one** value can occur, i.e., $P(Y = y_j) = 1$ for a given y_j, and $P(Y = y_i) = 0$ for all $i \neq j$ (**no uncertainty**), then $H(Y) = 0$

In particular, for binary random variables ($C = 2$):

$$H(Y) \in [0, \log_2 2] = [0, 1] \ .$$

Evaluating attributes' discriminant capability

Similarly, the **conditional entropy** of a random variable Y given another random variable X quantifies the amount of uncertainty about Y *when the value of* X *is known*, and is defined in terms of the **conditional** probability density function $P(Y|X)$:

$$H(Y|X) = \sum_{v} P(X = v) H(Y|X = v) \ ,$$

where the entropy $H(Y|X = v)$ is in turn defined as:

$$H(Y|X = v) = -\sum_{i=1}^{C} P(Y = y_i|X = v) \log_2 P(Y = y_i|X = v) \ .$$

Evaluating attributes' discriminant capability

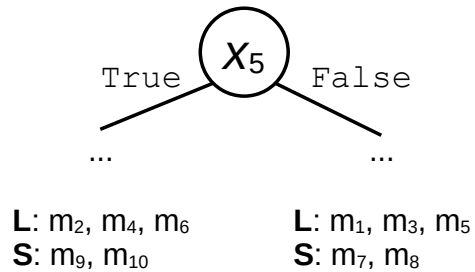
It is now easy to see that the **conditional entropy** of $P(Y|X)$ can also be used to measure the discriminant capability of an attribute x, since it exhibits the following properties:

- $H(Y|X) = 0$, if x perfectly splits the training set
- $H(Y|X) = \log_2 C$, if x uniformly splits the training set, for each of its values

Accordingly, the **conditional entropy** $H(Y|X)$ is used by ID3 to evaluate the discriminant capability of each attribute x, with respect to the training set $\mathcal{T}$, for the **root node** of a DT.

Discriminant capability of attributes: examples

As an example, consider again the attribute x_5:



The conditional entropy of $P(Y|X_5)$ is:

$$H(Y|X_5) = \sum_{v \in \{\texttt{True},\texttt{False}\}} P(X_5 = v) H(Y|X_5 = v) \ .$$

First, since we have 5 training examples with $x_5 = \texttt{True}$ and 5 examples with $x_5 = \texttt{False}$, $P(X_5)$ can be estimated as:

$$P(X_5 = \texttt{True}) = P(X_5 = \texttt{False}) = \frac{5}{10} = 0.5 \ .$$

Discriminant capability of attributes: examples

To compute $H(Y|X_5 = \texttt{True})$ and $H(Y|X_5 = \texttt{False})$, note first that there are 3 legitimate and 2 spam e-mails in $\mathcal{T}$ for both $x_5 = \texttt{True}$ and $x_5 = \texttt{False}$; therefore:

$$\begin{aligned} P(Y = \texttt{L}|X_5 = \texttt{True}) &= 3/5 = 0.6 \ , \\ P(Y = \texttt{S}|X_5 = \texttt{True}) &= 2/5 = 0.4 \ . \end{aligned}$$

It follows that:

$$H(Y|X_5 = \texttt{True}) = -0.6 \log_2 0.6 - 0.4 \log_2 0.4 \approx 0.97 \ .$$

A similar computation leads to:

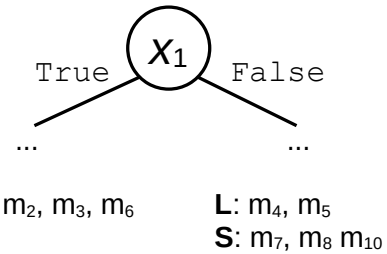
$$H(Y|X_5 = \texttt{False}) = -0.6 \log_2 0.6 - 0.4 \log_2 0.4 \approx 0.97 \ .$$

One finally obtains:

$$H(Y|X_5) \approx 0.5 * 0.97 + 0.5 * 0.97 = 0.97 \ .$$

Discriminant capability of attributes: examples

Consider now the attribute x_1:



The conditional entropy of $P(Y|X_1)$ is:

$$H(Y|X_1) = \sum_{v \in \{\texttt{True},\texttt{False}\}} P(X_1 = v) H(Y|X_1 = v)$$

Also in this case there are 5 training examples with $x_1 = \texttt{True}$ and 5 with $x_1 = \texttt{False}$, therefore:

$$P(X_1 = \texttt{True}) = P(X_1 = \texttt{False}) = \frac{5}{10} = 0.5 \ .$$

Discriminant capability of attributes: examples

To compute $H(Y|X_1 = \texttt{True})$, consider that there are 4 legitimate and 1 spam e-mail with $x_1 = \texttt{True}$, and therefore:

$$\begin{aligned} P(Y = \texttt{L}|X_1 = \texttt{True}) &= 4/5 = 0.8 \ , \\ P(Y = \texttt{S}|X_1 = \texttt{True}) &= 1/5 = 0.2 \ . \end{aligned}$$

It follows that:

$$H(Y|X_1 = \texttt{True}) = -0.8 \log_2 0.8 - 0.2 \log_2 0.2 \approx 0.72 \ .$$

To compute $H(Y|X_1 = \texttt{False})$, noting that there are 2 legitimate and 3 spam e-mail with $x_1 = \texttt{False}$, one obtains:

$$\begin{aligned} P(Y = \texttt{L}|X_1 = \texttt{False}) &= 2/5 = 0.4 \ , \\ P(Y = \texttt{S}|X_1 = \texttt{False}) &= 3/5 = 0.6 \ . \end{aligned}$$

Discriminant capability of attributes: examples

One then gets:

$$H(Y|X_1 = \texttt{False}) = -0.4 \log_2 0.4 - 0.6 \log_2 0.6 \approx 0.97 \ ,$$

which finally leads to:

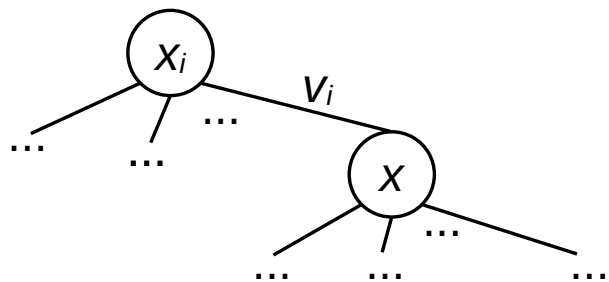
$$H(Y|X_1) \approx 0.5 * 0.72 + 0.5 * 0.97 = 0.845 \ .$$

Note that $H(Y|X_1) < H(Y|X_5)$; therefore, x_1 is **more discriminant** than x_5 for the **root node** of a DT.

Evaluating attributes' discriminant capability

The discriminant capability of attributes at inner nodes **different from the root** can be computed similarly.

For instance, assume that the attribute x_i has been chosen for the root node, and that a different attribute $x \neq x_i$ is being evaluated for the successor of the root corresponding to $x_i = v_i$:



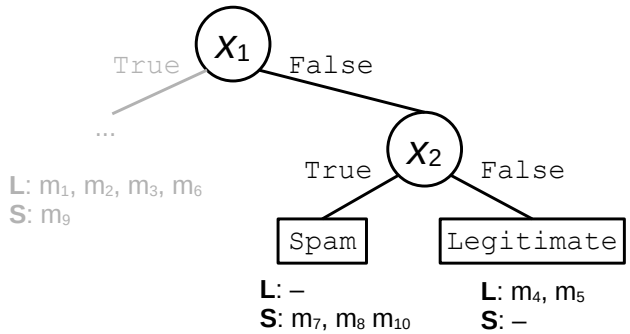
In this case only the training examples for which $x_i = v_i$ have to be considered.

Accordingly, the conditional entropy that has to be computed is:

$$H(Y|X, X_i = v_i) = \sum_v P(X = v|X_i = v_i) H(Y|X = v, X_i = v_i) \ .$$

Discriminant capability of attributes: examples

As an example, assume that x_1 has been chosen for the root node in the previous spam filtering example, and consider the discriminant capability of x_2 on the edge corresponding to $x_1 = \texttt{False}$:



In this case only $\texttt{m}_4$, $\texttt{m}_5$, $\texttt{m}_7$, $\texttt{m}_8$ and $\texttt{m}_{10}$ have to be considered, i.e., 2 legitimate and 3 spam e-mails.

The corresponding conditional entropy is:

$$H(Y|X_2, X_1 = \texttt{False}) = \sum_{v \in \{\texttt{True},\texttt{False}\}} P(X_2 = v|X_1 = \texttt{False}) H(Y|X_2 = v, X_1 = \texttt{False}) \ .$$

Discriminant capability of attributes: examples

Proceeding similarly as above, note first that:

$$\begin{aligned} P(X_2 = \texttt{True} | X_1 = \texttt{False}) &= 3/5 = 0.6 \ , \\ P(X_2 = \texttt{False} | X_1 = \texttt{False}) &= 2/5 = 0.4 \ . \end{aligned}$$

Since x_2 perfectly splits the considered training examples, one gets:

$$\begin{aligned} P(Y = \texttt{L} | X_2 = \texttt{True}, X_1 = \texttt{False}) &= 0 \ , \\ P(Y = \texttt{S} | X_2 = \texttt{True}, X_1 = \texttt{False}) &= 1 \ , \end{aligned}$$

and therefore $H(Y | X_2 = \texttt{True}, X_1 = \texttt{False}) = 0$.

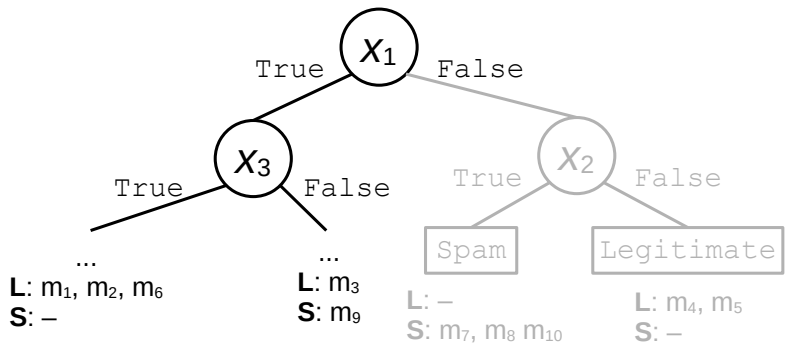
Similarly, $H(Y | X_2 = \texttt{False}, X_1 = \texttt{False}) = 0$.

It easily follows that the conditional entropy of x_2, **in this node**, equals 0:

$$H(Y | X_2, X_1 = \texttt{False}) = 0.6 \times 0 + 0.4 \times 0 = 0 \ .$$

Discriminant capability of attributes: examples

As a final example, consider the discriminant capability of x_3 along the edge corresponding to $x_1 = \texttt{True}$:



The corresponding conditional entropy is:

$$
\begin{aligned}
& H(Y|X_3, X_1 = \texttt{True}) \\
&= \sum_{v \in \{\texttt{True},\texttt{False}\}} P(X_3 = v|X_1 = \texttt{True}) H(Y|X_3 = v, X_1 = \texttt{True}) \ .
\end{aligned}
$$

Discriminant capability of attributes: examples

Proceeding as above:

$$\begin{aligned} P(X_3 = \texttt{True} | X_1 = \texttt{True}) &= 3/5 = 0.6 \ , \\ P(X_3 = \texttt{False} | X_1 = \texttt{True}) &= 2/5 = 0.4 \ . \end{aligned}$$

Considering the split corresponding to $x_3 = \texttt{True}$:

$$\begin{aligned} P(Y = \texttt{L} | X_3 = \texttt{True}, X_1 = \texttt{True}) = 1 \ , \\ P(Y = \texttt{S} | X_3 = \texttt{True}, X_1 = \texttt{True}) = 0 \ , \end{aligned}$$

and therefore $H(Y | X_3 = \texttt{True}, X_1 = \texttt{True}) = 0$.

Instead, for $x_3 = \texttt{False}$ we get two training examples uniformly distributed among the two classes, which implies:

$$H(Y | X_3 = \texttt{False}, X_1 = \texttt{True}) = 1 \ .$$

It follows that:

$$H(Y | X_3, X_1 = \texttt{True}) = 0.6 \times 0 + 0.4 \times 1 = 0.4 \ .$$

Evaluating attributes' discriminant capability

To sum up, to evaluate the discriminant capability of any attribute x associated with any inner node `n` of a DT, ID3 computes the conditional entropy of the probability distribution function $P(Y|X,\ldots)$

- conditioned, beside X, on the values of **all** the other attributes (if any) in the same branch
- considering only the training examples that arrive at node `n`

The **lower** the conditional entropy, the **higher** the discriminant capability. Accordingly, the attribute with the **lowest** conditional entropy is chosen.

Evaluating attributes' discriminant capability

Other measures have also been adopted by other learning algorithms, such as the **Gini index**, a statistical measure originally proposed by the Italian statistician Corrado Gini (1884–1965) to evaluate the degree of variation represented in a set of values, which is used especially in analysing income inequality.

Evaluating attributes' discriminant capability

According to Shannon's **Information Theory**, the entropy $H(Y)$ of a random variable Y can also be seen as the expected amount of "missing" **information** on its value (measured in *bits*).

Therefore, for DTs the entropy of the class label $H(Y)$ can be seen as representing the "missing" information that is needed to answer the question: "*What is the class label of a random instance?*"

Similarly, the conditional entropy of a given attribute x associated to the root node, $H(Y|X)$, is the amount of information that is needed to answer the same question, **after** observing the value of X.

Accordingly, choosing the attribute with the lowest conditional entropy amounts to minimising the "missing information", i.e., to maximise the **information gain** $H(Y) - H(Y|X)$.

The information gain quantifies how much information we have gained after a split.

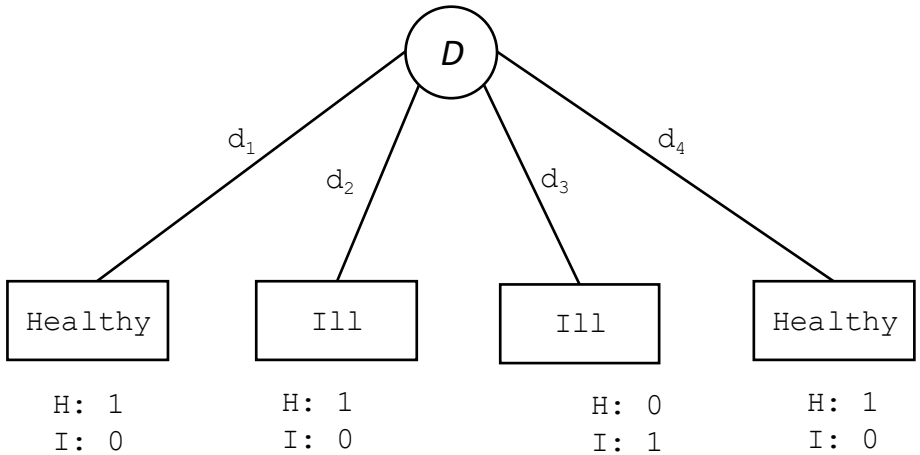
Evaluating attributes' discriminant capability

The conditional entropy turns out to be ineffective as a measure of discriminant capability for DTs, if an attribute is **irrelevant** to the class or has many values.

Consider for instance the problem of predicting whether a given person has a particular disease ($\mathcal{Y} = \{\texttt{healthy}, \texttt{ill}\}$), based on the outcome of some medical examination (e.g., blood test) and other information.

The **birth date** d is likely to be **irrelevant** to the health status; however, if it is included among the attributes, it is likely to exhibit a **different** value for each person in the training set.

Evaluating attributes' discriminant capability



$$H(Y|X) = \sum_{1}^{4} \frac{1}{4}\, 0 = 0.$$

Its conditional entropy $H(Y|D)$ is therefore likely to be equal to 0, and thus it would be selected as the most discriminant attribute for the root node. This produces a consistent DT, which however is likely to have no generalisation capability at all.

Evaluating attributes' discriminant capability

To avoid the drawbacks of conditional entropy (information gain), ID3 actually uses a variant of it, named **gain ratio**, which is more robust to irrelevant attributes and to attributes with many values.

$$\frac{InformationGain}{SplitInfo}.$$

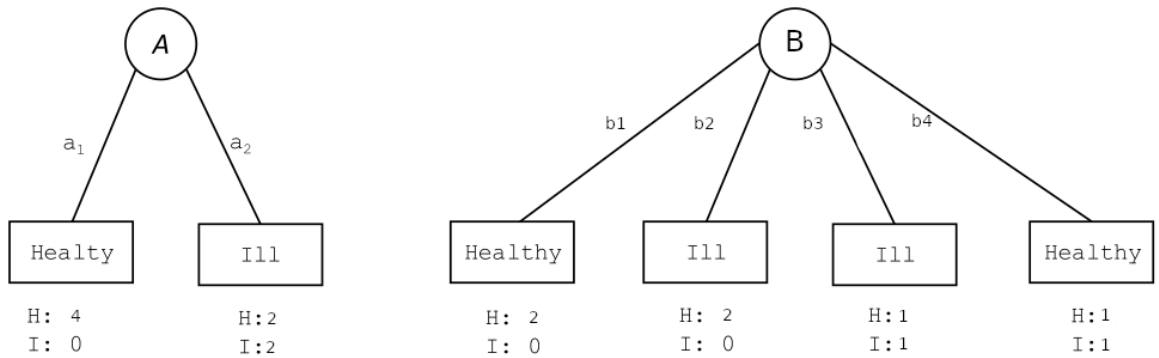
where:

$$SplitInfo = -\sum_{i}^{v} p_i log_2 p_i \;\; with \; p_i = \frac{N_i}{N}.$$

Where v is the number of values that the attribute can assume, N is the number of training samples arriving into the branch splitted by the attribute X, while N_i is the number of samples that arrive into branch edge created by the i-th value of X. The split Information is higher for attributes with more values.

Evaluating attributes' discriminant capability

Let's consider these two attributes for which the samples are equally distributed between the different values they can assume:



For the attribute A the Split Information is:

$$-2\ \frac{4}{8} log_2 \frac{4}{8} = 1.$$

For the attribute B the Split Information is:

$$-4\ \frac{2}{8} log_2 \frac{2}{8} = 2.$$

Evaluating attributes' discriminant capability

You can see the Split Information as the entropy of the considered attribute when we consider only the samples arriving into that branch.

For instance, the gain ratio of an attribute x at the root node of a DT is defined as:

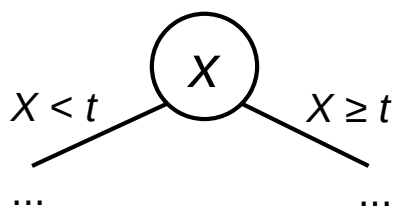
$$\frac{H(Y) - H(Y|X)}{H(X)} .$$

Extensions of ID3 to numerical attributes

For a numerical attribute x whose domain is a **finite** subset of integers, $\mathcal{X} = \{x_1, \ldots, x_n\} \subset \mathbb{I}$ (e.g., the number of edge pixels in an image), adding an edge for each of its values would be impractical for large n, and may also lead to over-fitting.

For numerical attributes, either integer or real, whose domain $\mathcal{X}$ is **infinite** (e.g., the frequency of a term in an e-mail), this is not even possible.

The solution used by ID3 and by other DT learning algorithms is to always use a **binary** split, i.e., subdividing $\mathcal{X}$ into **two** subsets corresponding to the tests $\mathcal{X}_{\text{lower}} = \{x \in \mathcal{X} : x < t\}$ and $\mathcal{X}_{\text{upper}} = \{x \in \mathcal{X} : x \geq t\}$, for a given threshold t:



Extensions of ID3 to numerical attributes

The threshold t has to be chosen by the learning algorithm.

To limit the number of threshold values to choose from, ID3 takes into account the different values of the attribute x among the training examples that reach the considered node.

Assuming that among such examples x takes q different values $x_{(1)} < x_{(2)} \ldots < x_{(q)}$, the following $q-1$ values of t are considered:

$$t_k = \frac{x_{(k+1)} + x_{(k)}}{2}, \quad k = 1, \ldots, q-1 \ .$$

Each of these values produces a **different** split of the training examples of interest, according to the tests $X < t_k$ and $X \geq t_k$: among them ID3 chooses the one corresponding to the **minimum** conditional entropy.

Decision regions for numerical attributes

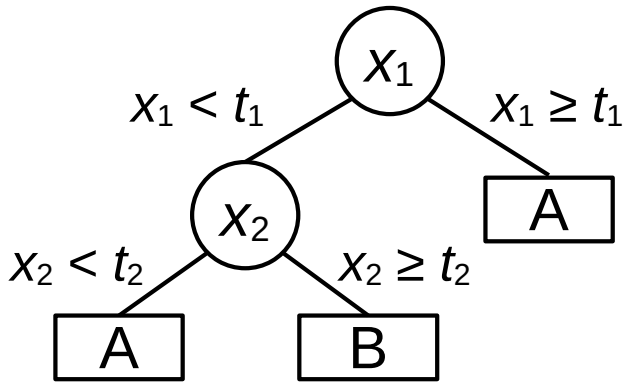
Any trained classifier can be seen as a function mapping from the attribute space to the label space, $h : \mathcal{X} \mapsto \mathcal{Y}$.

The regions of the attribute space $\mathcal{X}$ corresponding to the different class labels are named **decision regions**.

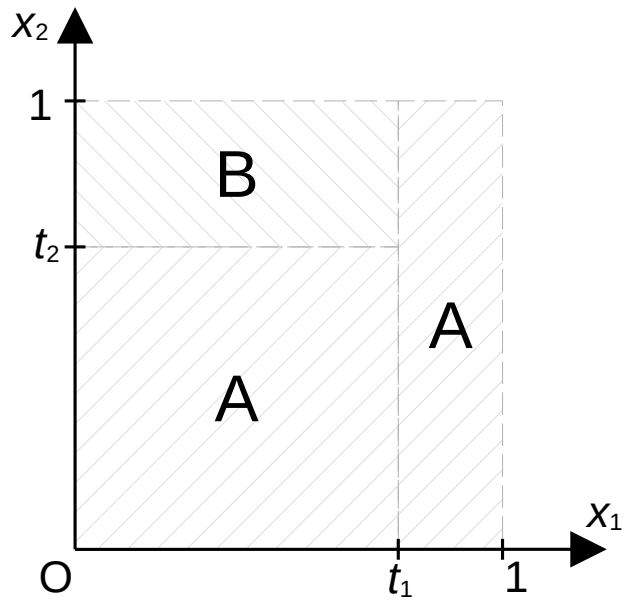
If the number d of attributes is not larger than 3, the decision regions of a classifier can be plotted in a d-dimensional graph.

Decision regions for numerical attributes

For instance, consider the DT on the right for a generic classification problem with labels `A` and `B`, with $d = 2$ numerical attributes x_1 and x_2 both ranging in $[0, 1]$, with binary splits produced by ID3.



The corresponding decision regions are shown in the plot on the right.



This example points out that the boundaries of the decision regions of a DT are made up of **lines parallel to the axes of the attribute space**.

Over-fitting in Decision Trees

Despite the choice of the most discriminant attribute for each inner node allows ID3 to build "small enough" DTs, the consistency requirement can nevertheless cause some degree of **over-fitting**.

This often happens when a too **deep** branch needs to be built to correctly classify a **small** number of training examples.
In this case the corresponding classification rule (leaf) is supported by **only a few** examples, and is therefore less likely represent a general characteristic of one of the classes, i.e., to **generalise** to unseen examples.

Mitigating over-fitting: Decision Tree *pruning*

To mitigate over-fitting, DT learning algorithms seek a **trade-off** between DT size and consistency: they avoid building deep branches devoted to a small number of examples, at the expense of consistency.

This can be achieved in two different ways:

- **offline**: after a consistent DT is built by a learning algorithm (e.g., ID3), it is "pruned" by removing sub-trees according to some criterion (hence the name **pruning**), and replacing them with a leaf
- **online**, by stopping the construction of a DT branch when some criterion is satisfied, and adding a leaf

In both cases, the leaf is usually associated with the label of the **majority** class among the training examples that reach it, to minimise the number of misclassifications in the training set.

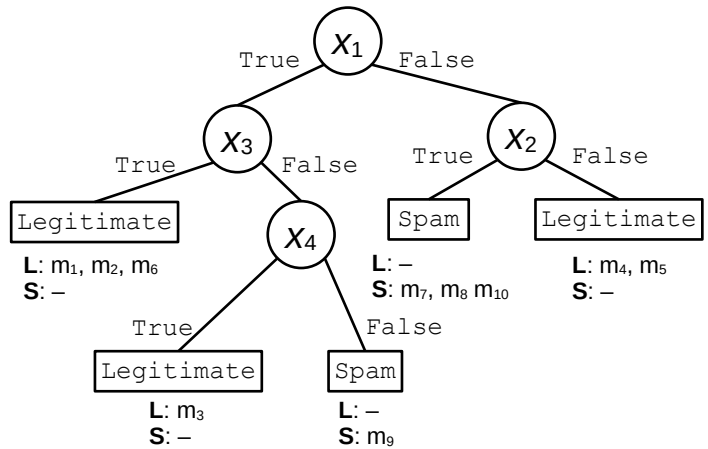
Mitigating over-fitting: Decision Tree *pruning*

Some possible pruning criteria (also applicable in the online fashion):

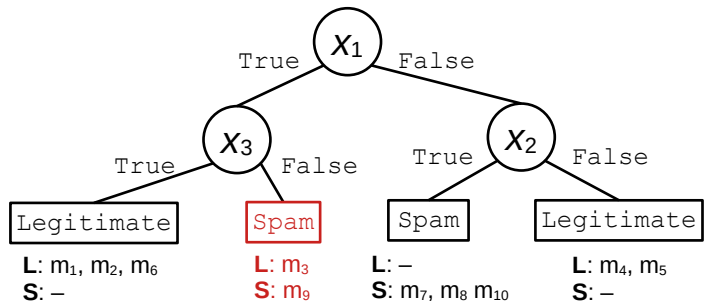
- setting a maximum tree depth
- the number of training examples reaching a node is lower than a predefined value
- the value of the splitting criterion (e.g., the gain ratio or the Gini index) of the most discriminant attribute is below or exceeds (depending on the considered metric) a predefined threshold (i.e., *the best attribute has a too low discriminant capability*, which may require a too high number of subsequent splittings to reach consistency)

Mitigating over-fitting: Decision Tree *pruning*

As an example, consider the DT built by ID3 for the spam filtering problem considered above, for a training set of ten e-mails:



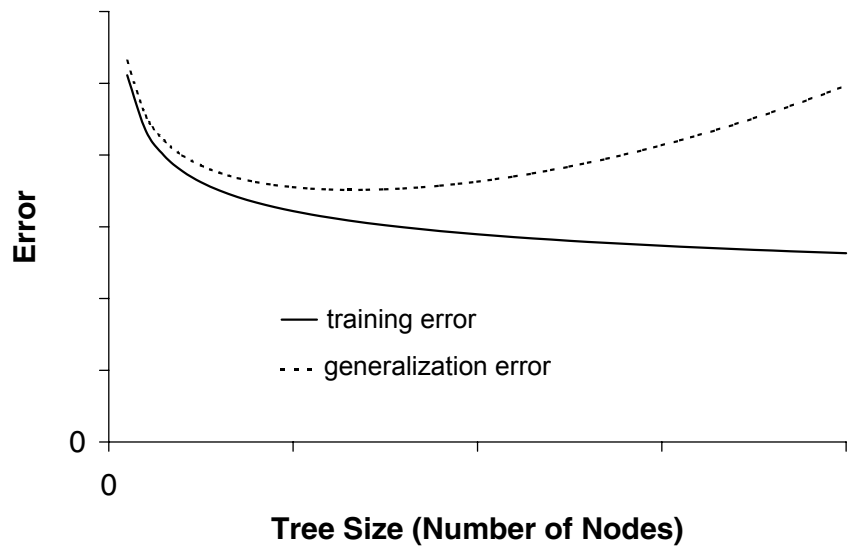
If the pruning criterion is to remove **subtrees** reached by **less than 3** training examples, the sub-tree whose root node is associated with x_4 should be removed and replaced with a leaf. Since it is reached by one legitimate and one spam e-mail, the class label can only be chosen randomly, e.g., `Spam`:



Over-fitting in Decision Trees

The **over-fitting** phenomenon can be observed in DTs by evaluating the number of misclassified training examples and an estimate of the generalisation capability, evaluated, e.g., using a **different** set of labelled examples (see below), as a function of DT size, for a **fixed** training set, and for different values of the pruning threshold.

A typical behaviour is represented in the figure below: the training error keeps decreasing as DT size increases, whereas the generalisation error initially decreases, then starts **increasing** at some point.



adapted from: L. Rokach and O. Maimon, *Data Mining with Decision Trees – Theory and Applications*, 2nd Ed., WorldScientific, 2014