BIM488 Introduction to Pattern Recognition

Classification Algorithms – Part III

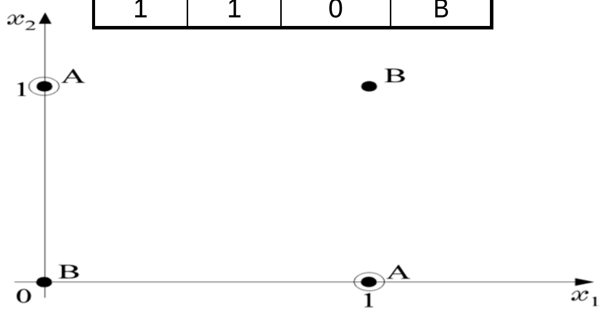
Outline

- Introduction
- Decision Trees

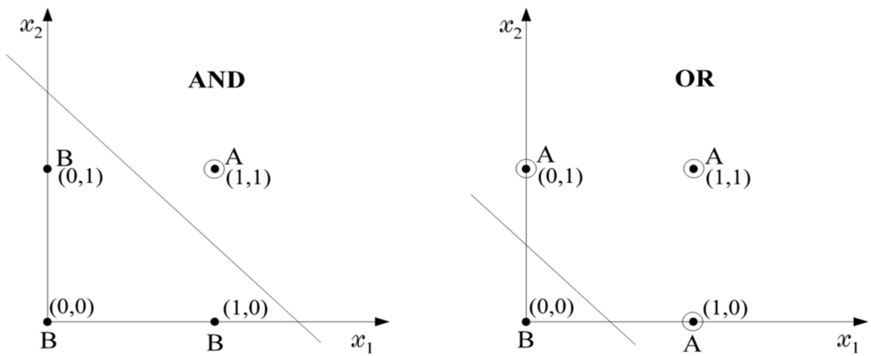
Introduction

❖ The XOR problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B



❖ There is no single line (hyperplane) that separates class A from class B. On the contrary, AND and OR operations are linearly separable problems



- There exist many types of nonlinear classifiers
  - Multi-layer neural networks
  - Support vector machines (nonlinear case)
  - Decision trees
  - ...
- We will particularly focus on decision trees in this course as a nonlinear classifier.

Decision Trees

Decision Trees are one of the widely used non-linear classifiers. They are multistage decision systems, in which classes are sequentially rejected, until a finally accepted class is reached. To this end:

- The feature space is split into **unique** regions in a sequential manner.

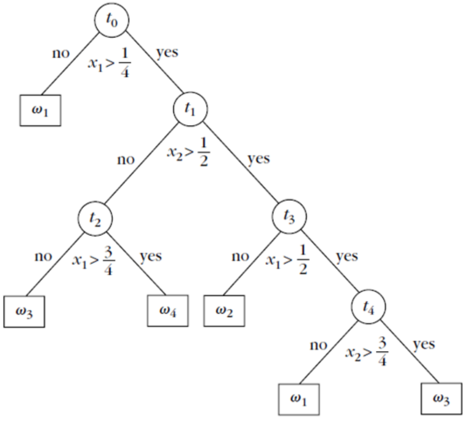
- Upon the arrival of a feature vector, sequential decisions, assigning features to specific regions, are performed along a path of nodes of an appropriately constructed tree.

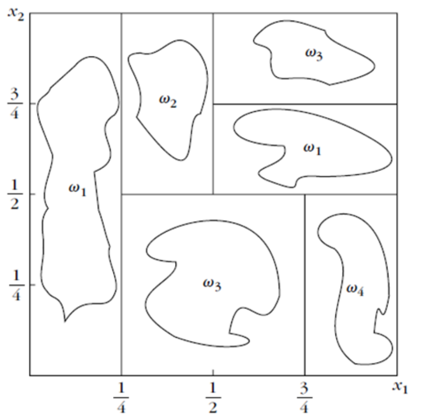
- The sequence of decisions is applied to individual features, and the queries performed in each node are of the type:

$$\text{is feature } x_i \leq \alpha \text{ ?}$$

where α is a pre-chosen (during training) threshold.

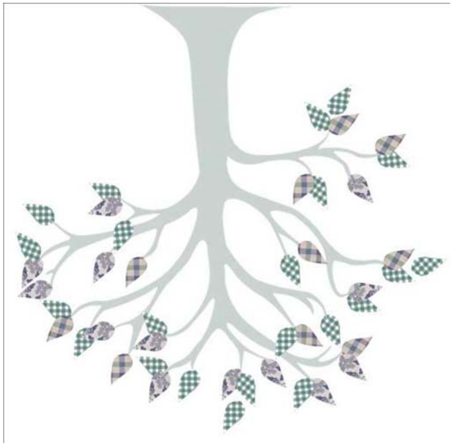
- The figures below are such examples. This type of trees is known as Ordinary Binary Classification Trees (OBCT). The decision hyperplanes, splitting the space into regions, are parallel to the axis of the spaces. Other types of partition are also possible, yet less popular.





Elements of a decision tree:

- Root
- Nodes
- Leafs



- Design Elements that define a decision tree.
  - Each node, t, is associated with a subset $X_t \subseteq X$, where X is the training set. At each node, X_t is split into two (binary splits) disjoint descendant subsets $X_{t,Y}$ and $X_{t,N}$, where

$$X_{t,Y} \cap X_{t,N} = \emptyset$$
$$X_{t,Y} \cup X_{t,N} = X_t$$

$X_{t,Y}$ is the subset of X_t for which the answer to the query at node t is YES. $X_{t,N}$ is the subset corresponding to NO. The split is decided according to an adopted question (query).

- A splitting criterion must be adopted for the best split of X_t into $X_{t,Y}$ and $X_{t,N}$.

- A stop-splitting criterion must be adopted that controls the growth of the tree and a node is declared as terminal (leaf).

- A rule is required that assigns each (terminal) leaf to a class.

- **Set of Questions:** In OBCT trees the set of questions is of the type

$$\text{is } x_i \leq \alpha \text{ ?}$$

The choice of the specific x_i and the value of the threshold α, for each node t, are the results of searching, during training, among the features and a set of possible threshold values. The final combination is the one that results to the best value of a criterion.

- Splitting Criterion: The main idea behind splitting at each node is the resulting descendant subsets $X_{t,Y}$ and $X_{t,N}$ to be more class homogeneous compared to X_t. Thus the criterion must be in harmony with such a goal. A commonly used criterion is the node impurity:

$$I(t) = -\sum_{i=1}^{M} P(\omega_i \mid t) \log_2 P(\omega_t \mid t)$$

and
$$P(\omega_i \mid t) \approx \frac{N_t^i}{N_t}$$

where N_t^i is the number of data points in X_t that belong to class ω_i. The decrease in node impurity (expected reduction in entropy, called as information gain) is defined as:

$$\Delta I(t) = I(t) - \frac{N_{t,\mathrm{Y}}}{N_t} I(t_{\mathrm{Y}}) - \frac{N_{t,N}}{N_t} I(t_N)$$

- The goal is to choose the parameters in each node (feature and threshold) that result in a split with the highest decrease in impurity. (Highest Information Gain)
  We wants higher information gain and lower entropy value
- *Why highest decrease?*
- Observe that the highest value of $I(t)$ is achieved if all classes are equiprobable, i.e., X_t is the least homogenous.
  I(t) =0.5 log2(0.5) + 0.5 log2(0.5) = 1.0
- Observe that the lowest value of $I(t)$ is achieved if data at the node belongs to only one class, i.e., X_t is the most homogenous.
  I(t) =1 log2(1) + 0 log2(0) = 0.0

- Where should we stop splitting?

- Stop - splitting rule: Adopt a threshold T and stop splitting a node (i.e., assign it as a leaf), if the impurity decrease is less than T. That is, node t is "pure enough".

- Class Assignment Rule: Assign a leaf to a class ω_j , where:

$$j = \arg\max_i P(\omega_i \mid t)$$

➢ Summary of an OBCT algorithmic scheme:

- Begin with the root node, i.e., $X_t = X$
- For each new node t
  - ∗ For every feature $x_k, k = 1, 2, \ldots, l$
    - For every value $\alpha_{kn}, n = 1, 2, \ldots, N_{tk}$
      - Generate X_{tY} and X_{tN} according to the answer in the question: is $x_k(i) \leq \alpha_{kn},\ i = 1, 2, \ldots, N_t$
      - Compute the impurity decrease
    - End
    - Choose α_{kn_0} leading to the maximum decrease w.r. to x_k
  - ∗ End
  - ∗ Choose x_{k_0} and associated $\alpha_{k_0 n_0}$ leading to the overall maximum decrease of impurity
  - ∗ If stop-splitting rule is met declare node t as a leaf and designate it with a class label
  - ∗ If not, generate two descendant nodes t_Y and t_N with associated subsets X_{tY} and X_{tN}, depending on the answer to the question: is $x_{k_0} \leq \alpha_{k_0 n_0}$
- End

Example:

In a tree classification task, the set X_t, associated with node t, contains $N_t = 10$ vectors. Four of these belong to class ω_1, four to class ω_2, and two to class ω_3, in a three-class classification task. The node splitting results into two new subsets X_{tY}, with three vectors from ω_1, and one from ω_2, and X_{tN} with one vector from ω_1, three from ω_2, and two from ω_3. The goal is to compute the decrease in node impurity after splitting.
We have that

$$I(t) = -\frac{4}{10}\log_2\frac{4}{10} - \frac{4}{10}\log_2\frac{4}{10} - \frac{2}{10}\log_2\frac{2}{10} = 1.521$$

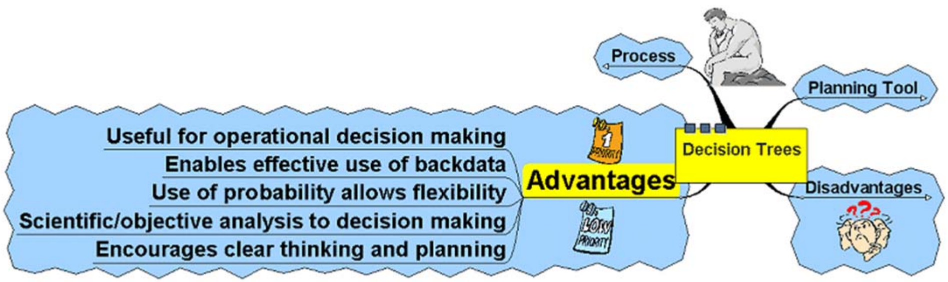
$$I(t_Y) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.815$$

$$I(t_N) = -\frac{1}{6}\log_2\frac{1}{6} - \frac{3}{6}\log_2\frac{3}{6} - \frac{2}{6}\log_2\frac{2}{6} = 1.472$$

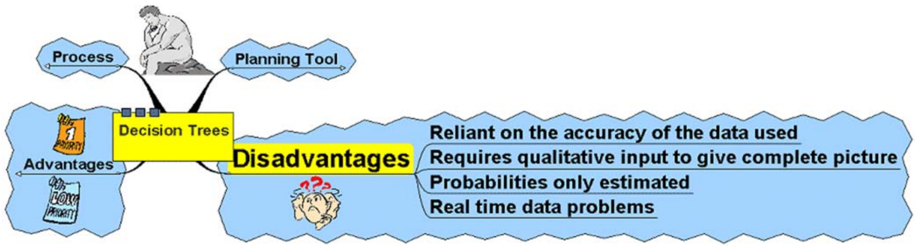
Hence, the impurity decrease after splitting is

$$\Delta I(t) = 1.521 - \frac{4}{10}(0.815) - \frac{6}{10}(1.472) = 0.315$$

Advantages


Process
Planning Tool
Useful for operational decision making
Enables effective use of backdata
Use of probability allows flexibility
Scientific/objective analysis to decision making
Encourages clear thinking and planning
Advantages
Decision Trees
Disadvantages

Disadvantages



Example:

Suppose we want to train a decision tree using the following instances:

Weekend (Examples)	Weather	Parents	Money	Decision (Category)
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
W3	Windy	Yes	Rich	Cinema
W4	Rainy	Yes	Poor	Cinema
W5	Rainy	No	Rich	Stay in
W6	Rainy	Yes	Poor	Cinema
W7	Windy	No	Poor	Cinema
W8	Windy	No	Rich	Shopping
W9	Windy	Yes	Rich	Cinema
W10	Sunny	No	Rich	Tennis

- The first thing we need to do is work out which attribute will be put into the node at the top of our tree: either weather, parents or money.

- To do this, we need to calculate:

$$Entropy(S) = -p_{cinema}\log_2(p_{cinema}) - p_{tennis}\log_2(p_{tennis}) - p_{shop}\log_2(p_{shop}) - p_{stay\_in}\log_2(p_{stay\_in})$$

$$= -(6/10) * \log_2(6/10) - (2/10) * \log_2(2/10) - (1/10) * \log_2(1/10) - (1/10) * \log_2(1/10)$$
$$= -(6/10) * -0.737 - (2/10) * -2.322 - (1/10) * -3.322 - (1/10) * -3.322$$
$$= 0.4422 + 0.4644 + 0.3322 + 0.3322 = 1.571$$

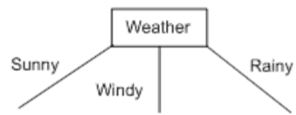
- and we need to determine the best of:

Gain(S, weather) = 1.571 - ($|S_{sun}|$/10)*Entropy(S_{sun}) - ($|S_{wind}|$/10)*Entropy(S_{wind}) - ($|S_{rain}|$/10)*Entropy(S_{rain})
= 1.571 - (0.3)*Entropy(S_{sun}) - (0.4)*Entropy(S_{wind}) - (0.3)*Entropy(S_{rain})
= 1.571 - (0.3)*(0.918) - (0.4)*(0.81125) - (0.3)*(0.918) = 0.70

Gain(S, parents) = 1.571 - ($|S_{yes}|$/10)*Entropy(S_{yes}) - ($|S_{no}|$/10)*Entropy(S_{no})
= 1.571 - (0.5) * 0 - (0.5) * 1.922 = 1.571 - 0.961 = 0.61

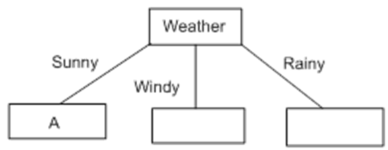
Gain(S, money) = 1.571 - ($|S_{rich}|$/10)*Entropy(S_{rich}) - ($|S_{poor}|$/10)*Entropy(S_{poor})
= 1.571 - (0.7) * (1.842) - (0.3) * 0 = 1.571 - 1.2894 = 0.2816

- This means that the first node in the decision tree will be the weather attribute. As an exercise, convince yourself why this scored (slightly) higher than the parents attribute - remember what entropy means and look at the way information gain is calculated.
- From the weather node, we draw a branch for the values that weather can take: sunny, windy and rainy:



- Now we look at the first branch. S_{sunny} = {W1, W2, W10}. This is not empty, so we do not put a default categorisation leaf node here. The categorisations of W1, W2 and W10 are Cinema, Tennis and Tennis respectively. As these are not all the same, we cannot put a categorisation leaf node here. Hence we put an attribute node here, which we will leave blank for the time being.

- Looking at the second branch, S_{windy} = {W3, W7, W8, W9}. Again, this is not empty, and they do not all belong to the same class, so we put an attribute node here, left blank for now. The same situation happens with the third branch, hence our amended tree looks like this:



- Now we have to fill in the choice of attribute A, which we know cannot be weather, because we've already removed that from the list of attributes to use. So, we need to calculate the values for Gain(S_{sunny}, parents) and Gain(S_{sunny}, money). Firstly, Entropy(S_{sunny}) = 0.918. Next, we set S to be S_{sunny} = {W1,W2,W10} (and, for this part of the branch, we will ignore all the other examples). In effect, we are interested only in this part of the table:

Weekend (Example)	Weather	Parents	Money	Decision (Category)
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
W10	Sunny	No	Rich	Tennis

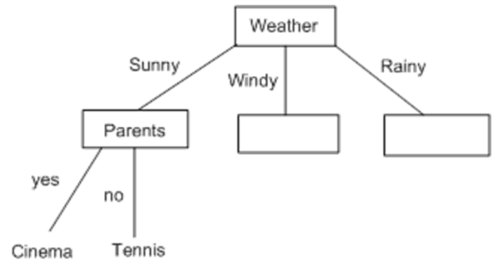
- Hence we can calculate:

$$\text{Gain}(S_{sunny}, \text{parents}) = 0.918 - (|S_{yes}|/|S|)*\text{Entropy}(S_{yes}) - (|S_{no}|/|S|)*\text{Entropy}(S_{no})$$
$$= 0.918 - (1/3)*0 - (2/3)*0 = 0.918$$

$$\text{Gain}(S_{sunny}, \text{money}) = 0.918 - (|S_{rich}|/|S|)*\text{Entropy}(S_{rich}) - (|S_{poor}|/|S|)*\text{Entropy}(S_{poor})$$
$$= 0.918 - (3/3)*0.918 - (0/3)*0 = 0.918 - 0.918 = 0$$

- Notice that Entropy(S_{yes}) and Entropy(S_{no}) were both zero, because S_{yes} contains examples which are all in the same category (cinema), and S_{no} similarly contains examples which are all in the same category (tennis). This should make it more obvious why we use information gain to choose attributes to put in nodes.

- Given our calculations, attribute A should be taken as parents. The two values from parents are yes and no, and we will draw a branch from the node for each of these. Remembering that we replaced the set S by the set S_{Sunny}, looking at S_{yes}, we see that the only example of this is W1. Hence, the branch for yes stops at a categorisation leaf, with the category being Cinema. Also, S_{no} contains W2 and W10, but these are in the same category (Tennis). Hence the branch for no ends here at a categorisation leaf. Hence our upgraded tree looks like this:



Finishing this tree off is left as an exercise !

Avoiding Overfitting

- As we discussed before, overfitting is a common problem in machine learning. Decision trees suffer from this, because they are trained to stop when they have perfectly classified all the training data, i.e., each branch is extended just far enough to correctly categorise the examples relevant to that branch. Many approaches to overcoming overfitting in decision trees have been attempted. These attempts fit into two types:

  - Stop growing the tree before it reaches perfection.
  - Allow the tree to fully grow, and then **post-prune** some of the branches from it.

- The second approach has been found to be more successful in practice.

Summary

- Introduction
- Decision Trees

References

- S. Theodoridis and K. Koutroumbas, *Pattern Recognition* (4th Edition), Academic Press, 2009.
- Decision Tree Learning, Lecture Notes of Course V231, Department of Computing, Imperial College, London.