Practical Lecture 1 - Perceptron and Decision Trees

Luis Sa-Couto[1] and Andreas Wichert[2]

INESC-ID, Instituto Superior Tecnico, Universidade de Lisboa
{luis.sa.couto,andreas.wichert}@tecnico.ulisboa.pt

1 Perceptron

The perceptron can be seen as an extremely simplified model of a biological neuron that is trained to fire when presented with elements of a given class.

More specifically, for an input vector $\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_d \end{pmatrix}^T$, the perceptron will, ideally, output $o = +1$ if $\mathbf{x}$ is an element of the learned class and $o = -1$ if it is not.

The knowledge that helps the perceptron decide if an element belongs to the class is stored in a vector of weights $\mathbf{w} = \begin{pmatrix} w_1 & w_2 & \cdots & w_d \end{pmatrix}^T$. For each feature x_i in the input vector, there is a weight w_i that decides the importance of that feature to recognize the class, more specifically:

- A very positive weight w_i indicates that if $\mathbf{x}$ has a large x_i than it is likely an instance of the class;
- A very negative weight w_i indicates that if $\mathbf{x}$ has a large x_i than it probably does not belong to the class;
- A w_i close to zero, indicates that x_i is not very relevant for the decision.

To implement this reasoning, the perceptron sums the input features weighted by their importance:

$$w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

Now, if the sum is large, then $\mathbf{x}$ is probably a member of the class. But how large? We need a threshold! Let's start by calling it T. If the sum is above T, the perceptron outputs $+1$. Otherwise, it outputs -1. More specifically, we want the perceptron to fire if:

$$perceptron\,(\mathbf{x}) = \begin{cases} +1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \geq T \\ -1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d < T \end{cases}$$

Which is the same as:

$$perceptron\,(\mathbf{x}) = \begin{cases} +1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d - T \geq 0 \\ -1 & w_1 x_1 + w_2 x_2 + \cdots + w_d x_d - T < 0 \end{cases}$$

The well-known sign function $sign\,(s) = \begin{cases} +1 & s \geq 0 \\ -1 & s < 0 \end{cases}$ can be used to write the perceptron computation in a more compact manner:

$$perceptron\left(\mathbf{x}\right) = sign\left(w_1 x_1 + w_2 x_2 + \cdots + w_d x_d - T\right)$$

In the literature, the threshold is usually treated as just another parameter refered to as the bias $w_0 = -T$, which leads to:

$$perceptron\left(\mathbf{x}\right) = sign\left(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d\right)$$

To write the model in an even more compact manner, we can add a dimension to $\mathbf{x}$ called x_0 that corresponds to the bias weight. Naturally, we must have $x_0 = 1$. Having said that, given the weight vector (augmented with the bias) $\mathbf{w} = \begin{pmatrix} w_0 & w_1 & w_2 & \cdots & w_d \end{pmatrix}^T$, we expand all input vectors as follows:

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_d \end{pmatrix}^T \rightarrow \begin{pmatrix} x_0 & x_1 & x_2 & \cdots & x_d \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & x_1 & x_2 & \cdots & x_d \end{pmatrix}^T$$

So, we can write the perceptron output as the sign of a dot product between input features and weights:

$$perceptron\left(\mathbf{x}\right) = sign\left(\sum_{i=0}^{d} w_i x_i\right) = sign\left(\mathbf{w} \cdot \mathbf{x}\right)$$

Now that we have covered how the perceptron makes decisions, we need to know how to learn the correct weights from training examples. To achieve that we sweep the data using the following procedure:

1. Initialize weights randomly
2. Get training example $\mathbf{x} \in \mathbb{R}^{\mathrm{d}}$ with target $t \in \{-1, +1\}$
3. Compute the perceptron output: $o = perceptron\left(\mathbf{x}\right)$
4. If it made a mistake: $o \neq t$, then for each feature x_i:
   (a) if output is positive $o = +1$ and it should be negative $t = -1$, subtract a fraction of x_i from w_i
   (b) if output is negative $o = -1$ and it should be positive $t = +1$, add a fraction of x_i to w_i
5. If there are more examples, go back to 2.

More specifically, if we define the size of the step (fraction of x_i) as $0 < \eta \leq 1$, we can write the learning rule for each feature as follows.

$$w_i = w_i + \eta\left(t - o\right) x_i$$

1) Consider the following linearly separable training set:

$$\left\{\mathbf{x}^1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mathbf{x}^2 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \mathbf{x}^3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathbf{x}^4 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}\right\}$$

$$\{t_1 = -1, t_2 = +1, t_3 = +1, t_4 = -1\}$$

a) Initialize all weights to one (including the bias). Use a learning rate of one for simplicity. Apply the perceptron learning algorithm until convergence.

Solution:

According to the question, we start with $\eta = 1$ and $\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T$.

Now, we take the first data point x^1 and augment it with a bias dedicated dimension:

$$\begin{pmatrix} x_0^1 & x_1^1 & x_2^1 \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$$

and compute the perceptron output for it:

$$o^1 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0\right) = +1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T + 1\left(-1 - 1\right)\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T = \begin{pmatrix} -1 & 1 & 1 \end{pmatrix}^T$$

We can then repeat the same logic for the next data point:

$$o^2 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2\right) = +1$$

As no mistake was made, we move to the next point:

$$o^3 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1\right) = +1$$

Again, no mistake was made, so we can move to the next point:

$$o^4 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot -1\right) = -1$$

No mistake was done again, which concludes our first sweep through the data (i.e. the first epoch). If we do another epoch, we see that the weights don't change. Thus, we have convergence.

---

b) Draw the separation hyperplane.

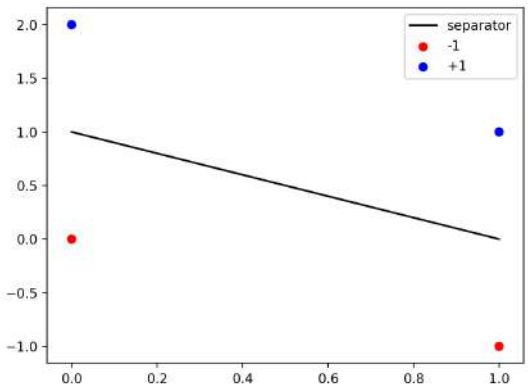
Solution:

Since we are working in two dimensions the weights define a separation line between the two classes. We get the equation for this line by checking the critical point where the decision changes from +1 to -1:

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$

$$\begin{aligned} w_0 x_0 + w_1 x_1 + w_2 x_2 &= 0 \\ (-1)1 + x_1 + x_2 &= 0 \\ x_2 &= -x_1 + 1 \end{aligned}$$

Having the line's equation, we can draw it in a plot with our data points to see the separation:



---

c) Does the perceptron converge on the first epoch if we change the weight initialization to zeros?

Solution:

According to the question, we start with $\eta = 1$ and $\mathbf{w} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$.

Now, we take the first data point x^1 and augment it with a bias dedicated dimension:

$$\begin{pmatrix} x_0^1 & x_1^1 & x_2^1 \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$$

and compute the perceptron output for it:

$$o^1 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(0 \cdot 1 + 0 \cdot 0 + 0 \cdot 0\right) = +1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T + 1\left(-1 - 1\right)\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T = \begin{pmatrix} -2 & 0 & 0 \end{pmatrix}^T$$

We can then repeat the same logic for the next data point:

$$o^2 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-2 \cdot 1 + 0 \cdot 0 + 0 \cdot 2\right) = -1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} -2 & 0 & 0 \end{pmatrix}^T + 1\left(1 - (-1)\right)\begin{pmatrix} 1 & 0 & 2 \end{pmatrix}^T = \begin{pmatrix} 0 & 0 & 4 \end{pmatrix}^T$$

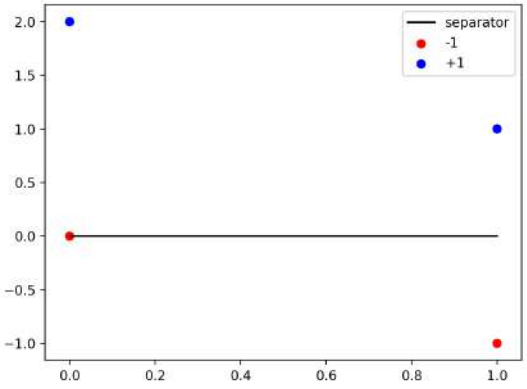
We can then repeat the same logic for the next data point:

$$o^3 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(0 \cdot 1 + 0 \cdot 1 + 4 \cdot 1\right) = +1$$

No mistake was made, so we can move to the next point:

$$o^4 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(0 \cdot 1 + 0 \cdot 1 + 4 \cdot -1\right) = +1$$

No mistake was made, so we finish the first epoch. We can plot the perceptron boundary and check that no convergence was achieved.



---

2) Consider the following linearly separable training set:

$$\left\{\mathbf{x}^1 = \begin{pmatrix}0\\0\\0\end{pmatrix}, \mathbf{x}^2 = \begin{pmatrix}0\\2\\1\end{pmatrix}, \mathbf{x}^3 = \begin{pmatrix}1\\1\\1\end{pmatrix}, \mathbf{x}^4 = \begin{pmatrix}1\\-1\\0\end{pmatrix}\right\}$$

$$\{t_1 = -1, t_2 = +1, t_3 = +1, t_4 = -1\}$$

a) Initialize all weights to one (including the bias). Use a learning rate of one for simplicity. Apply the perception learning algorithm for one epoch.

Solution:

According to the question, we start with $\eta = 1$ and $\mathbf{w} = \begin{pmatrix}1 & 1 & 1 & 1\end{pmatrix}^T$.

Now, we take the first data point x^1 and augment it with a bias dedicated dimension:

$$\begin{pmatrix}x_0^1 & x_1^1 & x_2^1 & x_3^1\end{pmatrix}^T \rightarrow \begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}^T$$

and compute the perceptron output for it:

$$o^1 = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0\right) = +1$$

Since the perceptron made a mistake, we use the output to compute an update:

$$\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^T + 1\,(-1-1)\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T = \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix}^T$$

We can then repeat the same logic for the next data point:

$$o^2 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 1) = +1$$

As no mistake was made, we move to the next point:

$$o^3 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1) = +1$$

Again, no mistake was made, so we can move to the next point:

$$o^4 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 0) = -1$$

No mistake was done again, which concludes the epoch.

---

b) For an additional epoch, do the weights change?

Solution:
We take the weights at the end of the first epoch:

$$\mathbf{w} = \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix}^T$$

and compute the perceptron output for all points:

$$o^1 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0) = -1$$

$$o^2 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 1) = +1$$

$$o^3 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1) = +1$$

$$o^4 = sign\,(\mathbf{w} \cdot \mathbf{x}) = sign\,(-1 \cdot 1 + 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 0) = -1$$

Since no mistakes occurred, we see that the perceptron converged.

---

c) What is the perceptron output for the query point $\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$?

Solution:
We start by augmenting the query with a bias dimension:

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \end{pmatrix}^T \rightarrow \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}^T$$

We take the converged weights:

$$\mathbf{w} = \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix}^T$$

And compute the perceptron output for the point:

$$o = sign\left(\mathbf{w} \cdot \mathbf{x}\right) = sign\left(-1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 1\right) = +1$$

We see that the point is on the boundary. So, because of the way we defined the sign function, the perceptron outputs +1, thus recognizing the point as a member of the learned class.

---

3) What happens if we replace the sign function by the step function?

$$\Theta\left(x\right) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Specifically, how would you change the learning rate to get the same results?

Solution:

Recall the lerning rule in question:

$$\mathbf{w} = \mathbf{w} + \eta\left(t - o\right)\mathbf{x}$$

The only term that depends on the perceptron output is the error $(y - o)$. With the sign function, the error is:

$$\delta_{sign} = t - o_{sign} = t - sign(\mathbf{w} \cdot \mathbf{x})$$

Which can be unfolded as:

$$\delta_{sign} = \begin{cases} +2 & t = +1, \mathbf{w} \cdot \mathbf{x} < 0 \\ -2 & t = -1, \mathbf{w} \cdot \mathbf{x} \geq 0 \end{cases}$$

Whereas with the step function, the error is:

$$\delta_{step} = t - o_{step} = t - \Theta(\mathbf{w} \cdot \mathbf{x})$$

Which can be unfolded as:

$$\delta_{step} = \begin{cases} +1 & t = +1, \mathbf{w} \cdot \mathbf{x} < 0 \\ -1 & t = -1, \mathbf{w} \cdot \mathbf{x} \geq 0 \end{cases}$$

Having said that, we notice that there is a relation between the two error terms $\delta_{sign} = 2\delta_{step}$.

So, since we had the update:

$$\mathbf{w} = \mathbf{w} + \eta_{sign}\delta_{sign}\mathbf{x}$$

If we replace the δ_{sign} by the error for the step function we get:

$$\mathbf{w} = \mathbf{w} + 2\eta_{sign}\delta_{step}\mathbf{x}$$

To get exactly the same update, we need to define a learning rate which is twice the one we used $\eta_{step} = 2\eta_{sign}$ and get:

$$\mathbf{w} = \mathbf{w} + \eta_{step}\delta_{step}\mathbf{x}$$

---

4) The perceptron can learn a relatively large number of functions. In this exercise, we focus on simple logical functions.

a) Show graphically that a perceptron can learn the logical NOT function. Give an example with specific weights.

Solution:

The NOT function receives as input a logical value $x \in \{-1, +1\}$ and outputs its logical negation $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

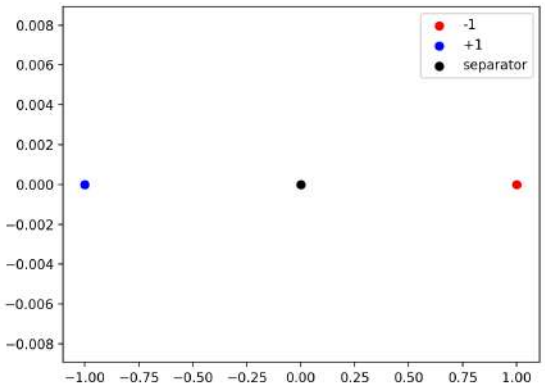
- For $x = -1$ the output is $t = +1$
- For $x = +1$ the output is $t = -1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output $(+1)$ can be separated from all points that require a negative output by an hyperplane.

Since we are working with 1 dimensional inputs, an hyperplane is, in this case, a point.

So, is there a point that accurately separates the points? Yes, any point between -1 and $+1$ will achieve this.

An example comes from setting the perceptron weights to zero:



b) Show graphically that a perceptron can learn the logical AND function for two inputs. Give an example with specific weights.

Solution:

The AND function receives as input a pair of logical values $\mathbf{x} \in \mathbb{R}^2$ and outputs its logical conjuntion $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

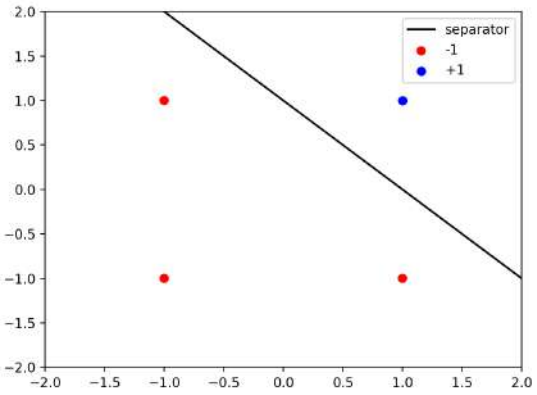
- For $\mathbf{x} = \begin{pmatrix} -1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} -1 & +1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} +1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} +1 & +1 \end{pmatrix}^T$ the output is $t = +1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output $(+1)$ can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2 dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

Yes, for instance $\mathbf{w} = \begin{pmatrix} -1 & 1 & 1 \end{pmatrix}^T$ achieves:



c) Show graphically that a perceptron can learn the logical OR function for two inputs. Give an example with specific weights.

Solution:

The OR function receives as input a pair of logical values $\mathbf{x} \in \mathbb{R}^2$ and outputs its logical disjunction $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

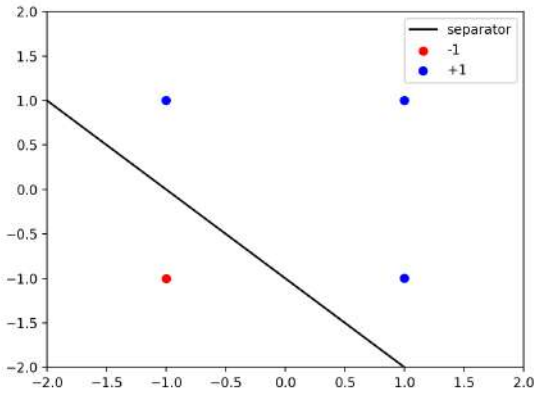
- For $\mathbf{x} = \begin{pmatrix} -1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} -1 & +1 \end{pmatrix}^T$ the output is $t = +1$
- For $\mathbf{x} = \begin{pmatrix} +1 & -1 \end{pmatrix}^T$ the output is $t = +1$
- For $\mathbf{x} = \begin{pmatrix} +1 & +1 \end{pmatrix}^T$ the output is $t = +1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output $(+1)$ can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2 dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

Yes, for instance $\mathbf{w} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T$ achieves:



---

d) Show graphically that a perceptron can not learn the logical XOR function for two inputs.

Solution:

The XOR function receives as input a pair of logical values $\mathbf{x} \in \mathbb{R}^2$ and outputs the exclusive disjunction $t \in \{-1, +1\}$. We can enumerate all possible inputs and their inputs:

- For $\mathbf{x} = \begin{pmatrix} -1 & -1 \end{pmatrix}^T$ the output is $t = -1$
- For $\mathbf{x} = \begin{pmatrix} -1 & +1 \end{pmatrix}^T$ the output is $t = +1$

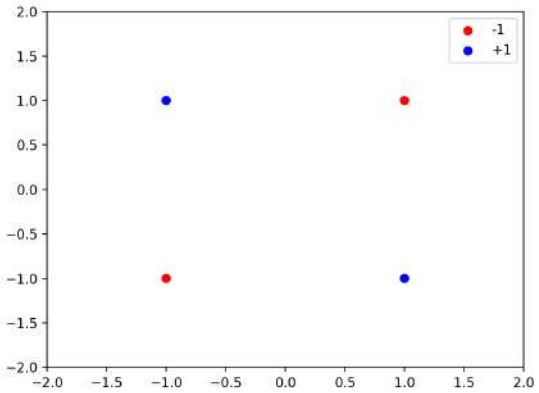
- For $\mathbf{x} = \begin{pmatrix} +1 & -1 \end{pmatrix}^T$ the output is $t = +1$
- For $\mathbf{x} = \begin{pmatrix} +1 & +1 \end{pmatrix}^T$ the output is $t = -1$

To show that a perceptron can learn a given function we need to show that all points that require a positive output (+1) can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2 dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

No... If we plot the points we see right away that no line can separate the positive instances from the negative ones:



2 Decision Trees

A decision tree represents a data table where each data point has features and a class. A node in the tree represents a test on a feature that partitions the original table in one table for each possible result of that test (see figure 1).

The tree is built by inserting these test nodes until we reach all leafs. Leafs are nodes where the table partition has either all elements of the same class or no more tests can be applied, for instance the rightmost node in the aforementioned figure. Once the full tree is built, to perform classification of a given point, we traverse the tree by applying each node's test until a leaf is reached (see figure 2).

The size of the tree depends on the order we choose for the tests. Different orders will result in different decision trees, all of which can be correct. So, how do we decide which one we want? On the one hand, we want a tree that correctly

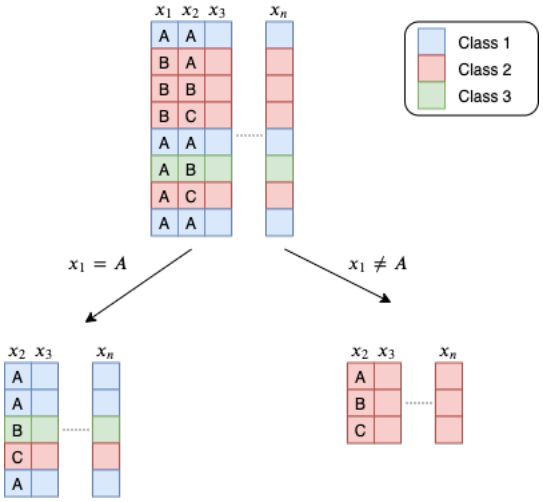


Fig. 1. A test node that creates two partitions.

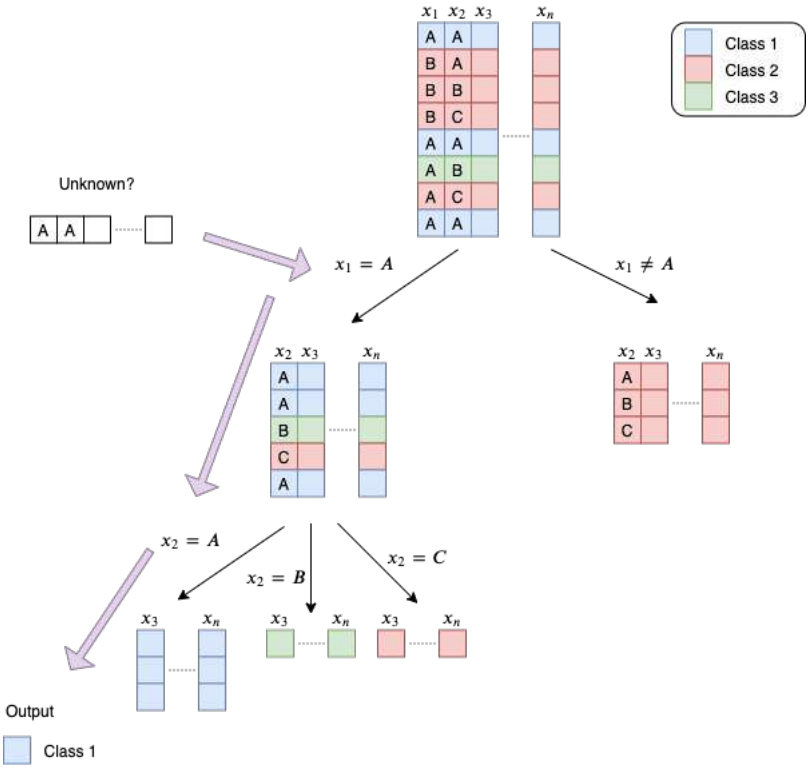


Fig. 2. Classifying a point with a decision tree.

captures the information present in the data table (i.e. classifies correctly its instances). On the other hand, we want a tree that classifies unseen examples correctly too. In that sense, a simple tree is preferrable because it is less likely to include unnecessary constraints that only appear true in this training set but are not true in general.

We could test all possible orders and choose the best tree but it grows extremely fast ($\Theta(n!)$). Since blind search won't do, we need a heuristic. A common procedure known as ID3 uses entropy as the heuristic.

Entropy is a function that receives a distribution and returns a value:

$$E(p_1, p_2, p_3, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

The higher the value the less deterministic the distribution is. More specifically, a high entropy tells us that the outcome of sampling that distribution is very hard to predict. Entropy of a given table is given by the entropy in the distribution of classes. How much entropy we need to get rid of to have perfect classification.

In ID3, a test node is a test on a specific feature, where one partition is created for each possible value. When building a tree, at a given moment, the entropy of that tree is the weighted average of the entropies of all partitions (see figure 3).

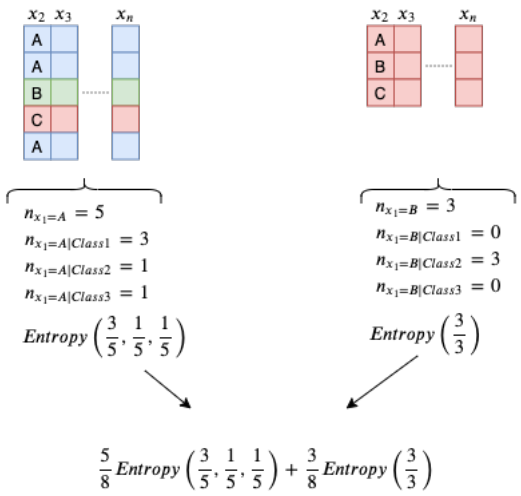


Fig. 3. Computing entropy from multiple partitions.

So, every time we want to choose the next attribute to test, we do:

– Compute the original entropy of the tree

- For each possible feature test we can add to the tree
  - compute the partition tables that are created each possible feature assignment
  - compute the total entropy of each partition
  - compute the new entropy by taking the weighted average of entropies of all partitions
  - compute the test's information gain by computing the difference between the original entropy and the new entropy
- Choose the feature that provides the highest information gain and add it to the tree

An alternative method we will see is CART where all tests have binary outputs. Concretely, instead of testing a feature and creating a branch for all its possible values, we test a feature value assignment $x_i = a$ and create two branches:

- A branch for instances where $x_i == a$;
- A branch for all reamining instances.

The test to place on the tree is chosen in a mechanic which is very similar to the one we used in ID3 but instead of entropy we will use impurity. Impurity is measured with the *Gini* index that also receives a distribution and outputs a value:

$$Gini\left(p_1, p_2, \ldots, p_n\right) = 1 - \sum_{i=1}^{n} p_i^2$$

1) Consider the following data table:

F_1	F_2	F_3	O
a	a	a	$+$
c	b	c	$+$
c	a	c	$+$
b	a	a	$-$
a	b	c	$-$
b	b	c	$-$

a) Determine the whole decision tree using ID3 (**information gain**), taking "O" as the target. Show all steps.

Solution:

Before anything, we need to compute the entropy we start with:

$$E_{start} = E\left(\frac{3}{3+3}, \frac{3}{3+3}\right) = 1bit$$

Let us test attribute F_1:

$$\begin{array}{ccc} F_1 = a & F_1 = b & F_1 = c \\ \downarrow & \downarrow & \downarrow \\ \#\{O=+\}=1 & \#\{O=+\}=0 & \#\{O=+\}=2 \\ \#\{O=-\}=1 & \#\{O=-\}=2 & \#\{O=-\}=0 \\ \downarrow & \downarrow & \downarrow \\ E\left(\frac{1}{1+1},\frac{1}{1+1}\right) & E\left(\frac{2}{2+0}\right) & E\left(\frac{2}{2+0}\right) \end{array}$$

Total entropy after partitioning by F_1 is equal to the weighted average of each partition's entropy: $E_{F_1} = \frac{2}{6}E\left(\frac{1}{1+1},\frac{1}{1+1}\right) + \frac{2}{6}E\left(\frac{2}{2+0}\right) + \frac{2}{6}E\left(\frac{2}{2+0}\right) \approx 0.33bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_1: $G(F_1) = E_{start} - E_{F_1} \approx 0.66bit$.

Let us test the next attribute, namely F_2:

$$\begin{array}{cc} F_2 = a & F_2 = b \\ \downarrow & \downarrow \\ \#\{O=+\}=2 & \#\{O=+\}=1 \\ \#\{O=-\}=1 & \#\{O=-\}=2 \\ \downarrow & \downarrow \\ E\left(\frac{2}{2+1},\frac{1}{2+1}\right) & E\left(\frac{1}{1+2},\frac{2}{1+2}\right) \end{array}$$

Total entropy after partitioning by F_2 is equal to the weighted average of each partition's entropy: $E_{F_2} = \frac{3}{6}E\left(\frac{2}{2+1},\frac{1}{2+1}\right) + \frac{3}{6}E\left(\frac{1}{1+2},\frac{2}{1+2}\right) = 0.9183bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_2: $G(F_2) = E_{start} - E_{F_2} = 0.0817bit$

For F_3:

$$\begin{array}{cc} F_3 = a & F_3 = c \\ \downarrow & \downarrow \\ \#\{O=+\}=1 & \#\{O=+\}=2 \\ \#\{O=-\}=1 & \#\{O=-\}=2 \\ \downarrow & \downarrow \\ E\left(\frac{1}{1+1},\frac{1}{1+1}\right) & E\left(\frac{2}{2+2},\frac{2}{2+2}\right) \end{array}$$

Total entropy after partitioning by F_3 is equal to the weighted average of each partition's entropy: $E_{F_3} = \frac{2}{6}E\left(\frac{1}{1+1},\frac{1}{1+1}\right) + \frac{4}{6}E\left(\frac{2}{2+2},\frac{2}{2+2}\right) = 1.0bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_3: $G(F_3) = E_{start} - E_{F_3} = 0bit$

Since F_1 provides the highest gain, we use it as the first node in the tree and get this break down of the dataset:

$F_1 = a$			$F_1 = b$			$F_1 = c$		
F_2	F_3	O	F_2	F_3	O	F_2	F_3	O
a	a	$+$	a	a	$-$	b	c	$+$
b	c	$-$	b	c	$-$	a	c	$+$
			Done!			*Done*!		

The first partition ($F_1 = a$) still has uncertainty. For that reason, we will repeat the same process to decide the next attribute to test. Before anything, we need to compute the entropy we start with:

$$E_{start} = E\left(\frac{1}{1+1}, \frac{1}{1+1}\right) = 1bit$$

Let us test the next attribute, namely F_2:

$$\begin{array}{cc} F_2 = a & F_2 = b \\ \downarrow & \downarrow \\ \#\{O = +\} = 1 & \#\{O = +\} = 0 \\ \#\{O = -\} = 0 & \#\{O = -\} = 1 \\ \downarrow & \downarrow \\ E\left(\frac{1}{1+0}\right) & E\left(\frac{1}{0+1}\right) \end{array}$$

Total entropy after partitioning by F_2 is equal to the weighted average of each partition's entropy: $E_{F_2} = \frac{1}{2}E\left(\frac{1}{1+0}\right) + \frac{1}{2}E\left(\frac{1}{0+1}\right) = 0bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_2: $G(F_2) = E_{start} - E_{F_2} = 1bit$

For F_3:

$$\begin{array}{cc} F_3 = a & F_3 = c \\ \downarrow & \downarrow \\ \#\{O = +\} = 1 & \#\{O = +\} = 0 \\ \#\{O = -\} = 0 & \#\{O = -\} = 1 \\ \downarrow & \downarrow \\ E\left(\frac{1}{1+0}\right) & E\left(\frac{1}{0+1}\right) \end{array}$$

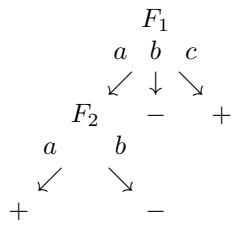
Total entropy after partitioning by F_3 is equal to the weighted average of each partition's entropy: $E_{F_3} = \frac{1}{2}E\left(\frac{1}{1+0}\right) + \frac{1}{2}E\left(\frac{1}{0+1}\right) = 0.0bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_3: $G(F_3) = E_{start} - E_{F_3} = 1bit$

Since both attributes have the same gain. we can choose either. Let us go with F_2 and get the following partioning:

$F_1 = a$				$F_1 = b$			$F_1 = c$		
$F_2 = a$		$F_2 = b$		F_2	F_3	O	F_2	F_3	O
F_3	O	F_3	O	a	a	$-$	b	c	$+$
a	$+$	c	$-$	b	c	$-$	a	c	$+$
Done!		*Done*!		*Done*!			*Done*!		

Since there is no more uncertainty, we needn't explore any further. Thus, we reach the final tree:



---

b) Repeat the same exercise but this time use CART (**Gini**).

Solution:

Before anything, we need to compute the impurity we start with:

$$Gini_{start} = Gini\left(\frac{3}{3+3}, \frac{3}{3+3}\right) = 1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2 = 0.50$$

Let us start with all value splits for attribute F_1:

First, we evaluate the split $\{a\}\, vs\, \{b,c\}$ which yields the following partition:

$F_1 = \{a\}$	$F_1 = \{b,c\}$
$\downarrow$	$\downarrow$
$\#\{O=+\} = 1$	$\#\{O=+\} = 2$
$\#\{O=-\} = 1$	$\#\{O=-\} = 2$
$\downarrow$	$\downarrow$
$Gini\left(\frac{1}{1+1}, \frac{1}{1+1}\right)$	$Gini\left(\frac{2}{2+2}, \frac{2}{2+2}\right)$

Which gives the weighted impurity:

$Gini_{\{a\}vs\{b,c\}}(F_1) = \frac{2}{2+4} Gini\left(\frac{1}{1+1}, \frac{1}{1+1}\right) + \frac{4}{2+4} Gini\left(\frac{2}{2+2}, \frac{2}{2+2}\right) = 0.5$

Which yields an impurity reduction of:

$\Delta Gini = Gini_{start} - Gini_{\{a\}vs\{b,c\}}(F_1) = 0.0$

Second, we evaluate the split $\{b\}\, vs\, \{a,c\}$ which yields the following partition:

$F_1 = \{b\}$	$F_1 = \{a,c\}$
$\downarrow$	$\downarrow$
$\#\{O=+\} = 0$	$\#\{O=+\} = 3$
$\#\{O=-\} = 2$	$\#\{O=-\} = 1$
$\downarrow$	$\downarrow$
$Gini\left(\frac{2}{0+2}\right)$	$Gini\left(\frac{3}{3+1}, \frac{1}{3+1}\right)$

Which gives the weighted impurity:

$Gini_{\{b\}vs\{a,c\}}(F_1) = \frac{2}{2+4} Gini\left(\frac{2}{0+2}\right) + \frac{4}{2+4} Gini\left(\frac{3}{3+1}, \frac{1}{3+1}\right) = 0.25$

Which yields an impurity reduction of:

$\Delta Gini = Gini_{start} - Gini_{\{b\}vs\{a,c\}}(F_1) = 0.25$

Finally, we evaluate the split $\{c\}\, vs\, \{a,b\}$ which yields the following partition:

$F_1 = \{c\}$	$F_1 = \{a, b\}$
$\downarrow$	$\downarrow$
$\#\{O = +\} = 2$	$\#\{O = +\} = 1$
$\#\{O = -\} = 0$	$\#\{O = -\} = 3$
$\downarrow$	$\downarrow$
$Gini\left(\frac{2}{2+0}\right)$	$Gini\left(\frac{1}{1+3}, \frac{3}{1+3}\right)$

Which gives the weighted impurity:

$$Gini_{\{c\}vs\{a,b\}}(F_1) = \frac{2}{2+4}Gini\left(\frac{2}{2+0}\right) + \frac{4}{2+4}Gini\left(\frac{1}{1+3}, \frac{3}{1+3}\right) = 0.25$$

Which yields an impurity reduction of:

$$\Delta Gini = Gini_{start} - Gini_{\{c\}vs\{a,b\}}(F_1) = 0.25$$

Now, we evaluate the next attribute:

$F_2 = \{a\}$	$F_2 = \{b\}$
$\downarrow$	$\downarrow$
$\#\{O = +\} = 2$	$\#\{O = +\} = 1$
$\#\{O = -\} = 1$	$\#\{O = -\} = 2$
$\downarrow$	$\downarrow$
$Gini\left(\frac{2}{2+1}, \frac{1}{2+1}\right)$	$Gini\left(\frac{1}{1+2}, \frac{2}{1+2}\right)$

Which gives the weighted impurity:

$$Gini_{\{a\}vs\{b\}}(F_2) = \frac{3}{3+3}Gini\left(\frac{2}{2+1}, \frac{1}{2+1}\right) + \frac{3}{3+3}Gini\left(\frac{1}{1+2}, \frac{2}{1+2}\right) = 0.4444$$

Which yields an impurity reduction of:

$$\Delta Gini = Gini_{start} - Gini_{\{a\}vs\{b\}}(F_2) = 0.0006$$

Finally, let us evaluate F_3:

$F_3 = \{a\}$	$F_3 = \{c\}$
$\downarrow$	$\downarrow$
$\#\{O = +\} = 1$	$\#\{O = +\} = 2$
$\#\{O = -\} = 1$	$\#\{O = -\} = 2$
$\downarrow$	$\downarrow$
$Gini\left(\frac{1}{1+1}, \frac{1}{1+1}\right)$	$Gini\left(\frac{2}{2+2}, \frac{2}{2+2}\right)$

Which gives the weighted impurity:

$$Gini_{\{a\}vs\{c\}}(F_3) = \frac{2}{2+4}Gini\left(\frac{1}{1+1}, \frac{1}{1+1}\right) + \frac{4}{2+4}Gini\left(\frac{2}{2+2}, \frac{2}{2+2}\right) = 0.5$$

Which yields an impurity reduction of:

$$\Delta Gini = Gini_{start} - Gini_{\{a\}vs\{c\}}(F_3) = 0$$

According to the algorithm, we choose the split that yields the maximum reduction, so, in this case it could be either $F_1 - \{b\}\, vs\, \{a, c\}$ or $F_1 - \{a, b\}\, vs\, \{c\}$. Let us pick the former.

From the chosen split we get two tables:

$F_1 = \{b\}$			$F_1 = \{a, c\}$		
F_2	F_3	O	F_2	F_3	O
a	a	$-$	a	a	$+$
b	c	$-$	b	c	$+$
			a	c	$+$
			b	c	$-$
Done!					

The first one is done since it has no impurity: $Gini_{start} = Gini\left(\frac{2}{2}\right) = 1 - \left(\frac{2}{2}\right)^2 = 0$.

For the second one, we need to repeat the whole process since we still have impurity: $Gini_{start} = Gini\left(\frac{3}{3+1}, \frac{1}{3+1}\right) = 0.3750$.

Let us start with F_2:

$F_2 = \{a\}$	$F_2 = \{b\}$
$\downarrow$	$\downarrow$
$\#\{O = +\} = 2$	$\#\{O = +\} = 1$
$\#\{O = -\} = 0$	$\#\{O = -\} = 1$
$\downarrow$	$\downarrow$
$Gini\left(\frac{2}{2+2}\right)$	$Gini\left(\frac{1}{1+1}, \frac{1}{1+1}\right)$

Which gives the weighted impurity:

$Gini_{\{a\}vs\{b\}}(F_2) = \frac{2}{2+2}Gini\left(\frac{2}{2+2}\right) + \frac{2}{2+2}Gini\left(\frac{1}{1+1}, \frac{1}{1+1}\right) = 0.25$

Which yields an impurity reduction of:

$\Delta Gini = Gini_{start} - Gini_{\{a\}vs\{b\}}(F_2) = 0.1250$

Finally, let us evaluate F_3:

$F_3 = \{a\}$	$F_3 = \{c\}$
$\downarrow$	$\downarrow$
$\#\{O = +\} = 1$	$\#\{O = +\} = 2$
$\#\{O = -\} = 0$	$\#\{O = -\} = 1$
$\downarrow$	$\downarrow$
$Gini\left(\frac{1}{1}\right)$	$Gini\left(\frac{2}{2+1}, \frac{1}{2+1}\right)$

Which gives the weighted impurity:

$Gini_{\{a\}vs\{c\}}(F_3) = \frac{1}{1+3}Gini\left(\frac{1}{1}\right) + \frac{3}{1+3}Gini\left(\frac{2}{2+1}, \frac{1}{2+1}\right) = 0.3333$

Which yields an impurity reduction of:

$\Delta Gini = Gini_{start} - Gini_{\{a\}vs\{c\}}(F_3) = 0.042$

So, again we choose the split that maximizes the reduction of impurity, which, in this case, is $F_2 - \{a\}\, vs\, \{b\}$. This choice will split the table in two:

$F_1 = \{b\}$

F_2	F_3	O
a	a	$-$
b	c	$-$

Done!

$F_1 = \{a, c\}$

$F_2 = \{a\}$

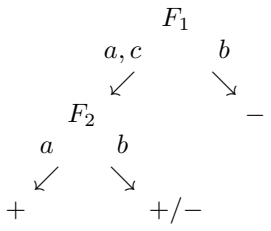
F_3	O
a	$+$
c	$+$

Done!

$F_2 = \{a\}$

F_3	O
c	$+$
c	$-$

The first table we get shows no impurtiy, so that branch is done. Whereas the second table has no split that allows to reduce impurity any further. For that reason, the algorithm stops with uncertainty on that branch. More specifically, it returns the following tree:



---

2) Consider the following data table:

F_1	F_2	F_3	O
d	a	b	m
c	a	b	n
c	a	a	y
d	a	a	y
c	b	a	f
c	b	b	f

a) Compute the first attribute to be tested using ID3.

Solution:

Before anything, we need to compute the entropy we start with:

$$E_{start} = E\left(\frac{\#\{O=m\}}{\sum_o \#\{O=o\}}, \frac{\#\{O=n\}}{\sum_o \#\{O=o\}}, \frac{\#\{O=y\}}{\sum_o \#\{O=o\}}, \frac{\#\{O=f\}}{\sum_o \#\{O=o\}}\right) =$$

$$= E\left(\frac{1}{6}, \frac{1}{6}, \frac{1}{3}, \frac{1}{3}\right) = 1.9183 bit$$

Let us test attribute F_1:

$$\begin{array}{cc} F_1 = d & F_1 = c \\ \downarrow & \downarrow \\ \#\{O=m\}=1 & \#\{O=m\}=0 \\ \#\{O=n\}=0 & \#\{O=n\}=1 \\ \#\{O=y\}=1 & \#\{O=y\}=1 \\ \#\{O=f\}=0 & \#\{O=f\}=2 \\ \downarrow & \downarrow \\ E\left(\frac{1}{1+1},\frac{1}{1+1}\right) & E\left(\frac{1}{1+1+2},\frac{1}{1+1+2},\frac{2}{1+1+2}\right) \end{array}$$

Total entropy after partitioning by F_1 is equal to the weighted average of each partition's entropy: $E_{F_1} = \frac{2}{6}E\left(\frac{1}{1+1},\frac{1}{1+1}\right) + \frac{4}{6}E\left(\frac{1}{1+1+2},\frac{1}{1+1+2},\frac{2}{1+1+2}\right) \approx 1.3333bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_1: $G(F_1) = E_{start} - E_{F_1} \approx 0.5850bit$.

Let us test the next attribute, namely F_2:

$$\begin{array}{cc} F_2 = a & F_2 = b \\ \downarrow & \downarrow \\ \#\{O=m\}=1 & \#\{O=m\}=0 \\ \#\{O=n\}=1 & \#\{O=n\}=0 \\ \#\{O=y\}=2 & \#\{O=y\}=0 \\ \#\{O=f\}=0 & \#\{O=f\}=2 \\ \downarrow & \downarrow \\ E\left(\frac{1}{1+1+2},\frac{1}{1+1+2},\frac{2}{1+1+2}\right) & E\left(\frac{2}{2}\right) \end{array}$$

Total entropy after partitioning by F_2 is equal to the weighted average of each partition's entropy: $E_{F_2} = \frac{4}{6}E\left(\frac{1}{1+1+2},\frac{1}{1+1+2},\frac{2}{1+1+2}\right) + \frac{2}{6}E\left(\frac{2}{2}\right) \approx 1.0bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_2: $G(F_2) = E_{start} - E_{F_2} \approx 0.9183bit$

For F_3:

$$\begin{array}{cc} F_3 = b & F_3 = a \\ \downarrow & \downarrow \\ \#\{O=m\}=1 & \#\{O=m\}=0 \\ \#\{O=n\}=1 & \#\{O=n\}=0 \\ \#\{O=y\}=0 & \#\{O=y\}=2 \\ \#\{O=f\}=1 & \#\{O=f\}=1 \\ \downarrow & \downarrow \\ E\left(\frac{1}{1+1+1},\frac{1}{1+1+1},\frac{1}{1+1+1}\right) & E\left(\frac{2}{2+1},\frac{1}{2+1}\right) \end{array}$$

Total entropy after partitioning by F_3 is equal to the weighted average of each partition's entropy: $E_{F_3} = \frac{3}{6}E\left(\frac{1}{1+1+1},\frac{1}{1+1+1},\frac{1}{1+1+1}\right) + \frac{3}{6}E\left(\frac{2}{2+1},\frac{1}{2+1}\right) \approx 1.2516bit$.

So, the information gain is given by subtracting the entropy at begining from the remaining entropy after F_3: $G(F_3) = E_{start} - E_{F_3} \approx 0.6667 bit$

Since F_2 provides the highest gain, we would use it as the first node in the tree.

---

b) Complete the tree started in the previous question. There is no need to perform all computations. What do you need to take into account?

Solution:

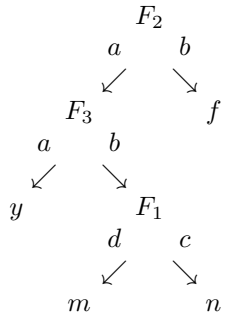
From last question we got that the root node should be a test on F_2, this yields the following partition:

$F_2 = a$			$F_2 = b$		
F_1	F_3	O	F_1	F_3	O
d	b	m	c	a	f
c	b	n	c	b	f
c	a	y	*Done*!		
d	a	y			

Analyzing the second table, we see that no more uncertainty exists, so that branch is completed. The first table still provides uncertainty, so we would have to decide between testing F_1 or F_3. Testing F_1 would give no advantage since both branche would end up undecided. Whereas testing F_3 allows us to separate the y's from the rest. So, we would get the follwoing partition:

$F_2 = a$				$F_2 = b$		
$F_3 = b$		$F_3 = a$		F_1	F_3	O
F_1	O	F_1	O	c	a	f
d	m	c	f	c	b	f
c	n	d	f	*Done*!		
		Done!				

With this test, two branches appear. On the one hand, for $F_3 = a$, uncertainty disappears, so it is done. On the other hand, for $F_3 = a$, we require a final test on F_1 to get rid of the final bit of uncertainty. With that test, we get the final tree:

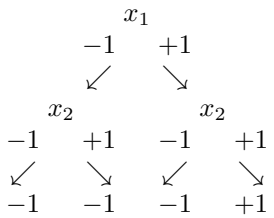


3) Show if a decision tree can learn the following logical functions and if so plot the corresponding decision boundaries.

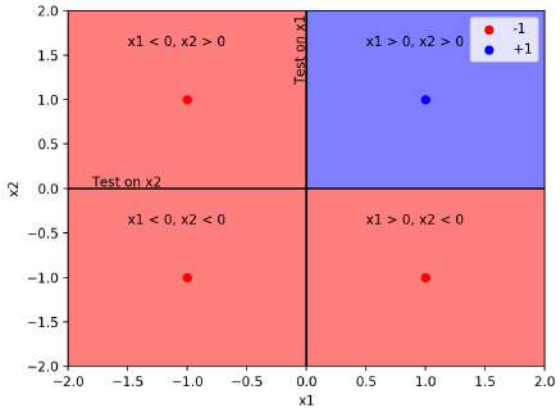
a) AND

Solution:

To show it is possible we only need to create a decision tree that solves the problem. The following does:



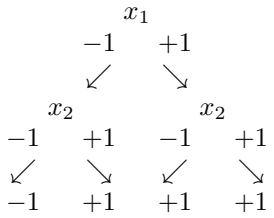
The corresponding decision boundaries are shown below.



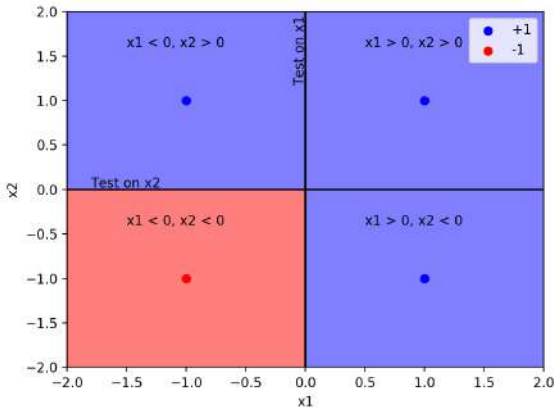
b) OR

Solution:

To show it is possible we only need to create a decision tree that solves the problem. The following does:



The corresponding decision boundaries are shown below.

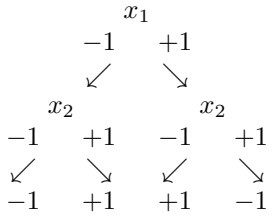


---

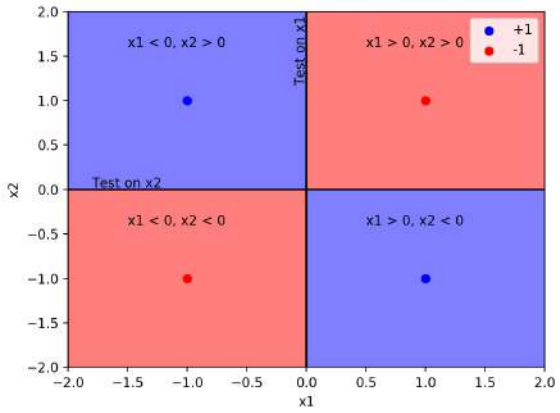
c) XOR

Solution:

To show it is possible we only need to create a decision tree that solves the problem. The following does:



The corresponding decision boundaries are shown below.



3 Thinking Questions

a) Taking into account the answers to the previous question reflect on how the decision tree can represent much more complex functions than the perceptron. Reflect also on how the size of the tree relates to the number of regions in which the space can be divided. Can you think why we want a good heuristic when building the tree? Why are usually smaller trees preferred?

b) Could you implement a statistical machine learning application that classifies a number into two classes, primes and non-primes? What is the main difference between this problem and the salmon and sea bass example from the lecture?