# *Tutorial 02 — Answers*

(Version 1.1)

1. (a) The algorithm for building a decision tree is:

   - if there are no more examples, return the plurality classification.
   - else-if examples are all classified the same, return the classification.
   - else-if there are no more attributes, return the plurality classification.
   - else pick the best attribute $A$ as root for the tree.
   - for each value of $A$
     - collect the examples which match that value of $A$.
     - use a recursive call to this procedure to build a subtree.

   Let's step through this algorithm for the example.

   Since we have examples and attributes, we start by picking the best attribute. Let's say this is $Lang$ (a lucky guess, see below).

   After applying $Lang$, we have three branches:

   - One for $Lang = Eng$ with examples $X_1$, $X_3$ and $X_9$.
   - One for $Lang = Sp$ with examples $X_2$, $X_4$, $X_5$, $X_6$ and $X_8$.
   - One for $Lang = Fr$ with examples $X_7$ and $X_{10}$.

   We now apply a recursive call of this procedure to each branch.

   For the branch for $Lang = Fr$, all the examples are labelled $N$, so there is no more work to do — according to WebMooviz if the language is French, I won't stream a film[1]

   For the branch for $Lang = Sp$, we have a mix of $Y$ and $N$ labelled examples. So we have to pick another attribute. Let's say we pick $Type$. Here we again have three branches:

   - One for $Type = Comedy$ with examples $X_2$ and $X_4$.
   - One for $Type = Drama$ with one example, $X_6$
   - One for $Type = Action$ with examples $X_5$ and $X_8$.

   For the branch for $Type = Comedy$, both the examples are labelled $Y$, so we are done.

   For the branch $Type = Drama$, we have only one example, so we are also done.

   For the branch $Type = Action$, we have two examples, one labelled $Y$ and one labelled $X$, so we will have to make another recursive call.

   When that call is complete (and to ruin any suspense I can reveal we will be able to classify the remaining examples using the attribute $New$), we will go back to consider the branch with $Lang = Eng$.

   Note that, for pedagogical purposes, I haven't followed the DTL algorithm exactly. that works strictly depth-first, making the recursive call on the first attribute value for which the computation hasn't bottomed out. So after selecting $Lang$ as an attribute, it would order the three values $EnG$, $Fr$ and $Sp$ somehow, then drill down to the bottom of the first branch in that order, then the second, ...

[1] This goes to show the dangers of learning from a small set of examples — I like quite a lot of French films.

Note also that unfolding the $Lang = Eng$ branch will give us a case where we need to use the plurality classification. If we first look at $New$, we will immediately classify $X_1$ and be left with $x_3$ and $X_9$, two $Drama$ films, one of which got a $Y$ and one of which got a $N$. With no more attributes to test, we can do no better than to return $Y$ or $N$ with a probability of $0.5$.

(b) We are choosing between the attributes $New$, $Lang$ and $Type$. The overall training set has six positive examples and four negative examples. So the total entropy of the set of examples is:

$$H(Goal) = B\left(\frac{p}{p+n}\right)$$

where $B(q)$ is the entropy of a random variable with probability $q$:

$$B(q) = -(qlog_2(q) + (1-q)log_2(1-q))$$

In this case:

$$H(Goal) = B\left(\frac{6}{10}\right) = 0.97$$

For any attribute $A$, we can compute the entropy after testing on $A$ as:

$$Remainder(A) = \sum_i \frac{p_i + n_i}{p+n} B\left(\frac{p_i}{p_i + n_i}\right)$$

where $p_i$ is the number of positive examples for the $i$th value of $A$. Then:

$$Gain(A) = H(Goal) - Remainder(A)$$

$Lang$ has values $Eng$, $Sp$ and $Fr$. $Eng$ has two positive and one negative example, $Sp$ has four positive and one negative example, and $Fr$ has no positive and two negative examples. Thus:

$$\begin{aligned}
Remainder(Lang) &= \frac{3}{10}B\left(\frac{2}{3}\right) + \frac{5}{10}B\left(\frac{4}{5}\right) + \frac{2}{10}B\left(\frac{0}{2}\right) \\
&= 0.636 \\
Remainder(Type) &= 0.951 \\
Remainder(New) &= 0.846
\end{aligned}$$

from which we see that $Lang$ is the best attribute to use as the root of the decision tree.

2. To use Gini Impurity in the decision about which attribute to select next in constructing a decision tree, we follow a process that is similar to that for using entropy:

```
For each attribute
    For each value
        Compute the Gini impurity of the attribute taking that value
    Compute the Gini impurity of the attribute

Pick the attribute with the lowest Gini Impurity
```

We will start with the attribute $Lang$, which has possible values $Eng$, $SP$ and $Fr$.

There are three examples such that $Lang = Eng$, $X_1$, $X_3$ and $X_9$. Two of these are positive, and 1 is negative. Recall that:

$$G(S) = 1 - \left(\left(\frac{p}{p+n}\right)^2 + \left(\frac{n}{p+n}\right)^2\right)$$

so,

$$\begin{aligned} G(Eng) &= 1 - \left(\left(\frac{2}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right) \\ &= 1 - (0.444 + 0.111) \\ &= 1 - 0.555 \\ &= 0.444 \end{aligned}$$

Similarly, $Lang = Sp$ has four positive examples and one negative example, so that:

$$\begin{aligned} G(Sp) &= 1 - \left(\left(\frac{4}{5}\right)^2 + \left(\frac{1}{5}\right)^2\right) \\ &= 1 - (0.64 + 0.04) \\ &= 1 - 0.68 \\ &= 0.32 \end{aligned}$$

Finally, $Lang = Fr$ has two negative examples, so:

$$\begin{aligned} G(Fr) &= 1 - \left(\left(\frac{0}{2}\right)^2 + \left(\frac{2}{2}\right)^2\right) \\ &= 1 - (0 + 1) \\ &= 0 \end{aligned}$$

Now we can calculate the Gini Impurity of $Lang$, which is the weighted sum of the Gini Impurities of the values of $Lang$:

$$G(S, A) = \sum_i \frac{|S_i|}{|S|} G(S_i)$$

Here $A$ is $Lang$, the $S_i$ are the sets of examples for each attribute value, and $S$ is the full set of values, which in the previous question we were calling $Goal$, so:

$$\begin{aligned} G(Goal, Lang) &= \frac{3}{10}G(Eng) + \frac{5}{10}G(Sp) + \frac{2}{10}G(Fr) \\ &= \frac{3}{10}0.444 + \frac{5}{10}0.32 + \frac{2}{10}0 \\ &= 0.293 \end{aligned}$$

We then do the same kind of calculation for $New$ and $Type$, and we find that:

$$\begin{aligned} G(Goal, New) &= 0.4 \\ G(Goal, Type) &= 0.46 \end{aligned}$$

So the attribute with the lowest Gini Impurity as the root of the tree is $Lang$, and so that is the one we should pick.

In this case, Gini Impurity picks the same attribute to use as entropy does. This is not the case in general.

3. (a) Broadly speaking, an $n$-dimensional dataset is linearly separable if it can be partitioned using a single decision surface with $n-1$ dimensions. Thus a linearly separable 2 dimensional dataset can be separated by a straight line (a one-dimensional decision surface).

   (b) Again I get to use the examples from the slides.

   Slide 81 gives the classic examples. (a) and (b) show sets of three points which can be separated from the fourth point by a single straight line (a one dimensional surface — the line can be specified as a function of either $x_1$ or $x_2$). Example (c) shows a dataset that is not linearly separable because there is no (straight) line that can be drawn to separate the two sets. We can, of course, draw an ellipse around two of the points, but that is not a one-dimensional — the equation of an ellipse in the plane of $x_1$ and $x_2$ is a function of both $x_1$ and $x_2$ and hence is two dimensional.

   (c) You can't build a classifier for a non-linearly separable dataset using a linear classifier. So, if an n-dimensional dataset is not linearly separable, linear regression (or a perceptron, or any other linear technique) will not be able to lear a good classifier.

   Such a dataset will need either something like a neural network (which fuses multiple linear classifiers to create something that will separate the data — for example (c) on slide 81 can be separated using two straight lines, and hence could potentially be handled by two linear classifiers) or a support vector machine (which uses the kernel trick to allow a linear classifier to work).

   Note that the last paragraph probably won't make sense for now. Don't worry, by the time the module is over, it will. Come back and read it then.

4. (a) The formula for batch gradient descent is:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j))$$
$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j))x_j$$

   To apply it we first need to calculate:

$$\sum_j (y_j - h_w(x_j))$$

   Think of this as:

$$\sum_j (y_j - h_w(x_j)) = \sum_j error_j^0$$

   where:

$$error_j^0 = y_j - h_w(x_j)$$
$$h_w(x_j) = w_0 + w_1 x_j$$

So, with $w_0$ and $w_1$ both 0, we have:

$$\begin{aligned} error_1^0 &= 1 - (w_0 + w_1 x_1) \\ &= 1 \\ error_2^0 &= 3 \\ error_3^0 &= 2 \\ error_4^0 &= 3 \\ error_5^0 &= 2.5 \end{aligned}$$

and:

$$\sum_j (y_j - h_w(x_j) = 11.5$$

we then run the update of $w_0$:

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j)) \\ w_0 &\leftarrow 0 + 0.01 \times 11.5 \\ w_0 &= 0.115 \end{aligned}$$

Then we need to calculate:

$$\sum_j (y_j - h_w(x_j) x_j$$

Think of this as:

$$\sum_j (y_j - h_w(x_j) x_j = \sum_j error_j^1$$

where:

$$\begin{aligned} error_j^1 &= (y_j - h_w(x_j)) x_j \\ h_w(x_j) &= w_0 + w_1 x_j \end{aligned}$$

So, with $w_0$ and $w_1$ both 0, we have:

$$\begin{aligned} error_1^1 &= (1 - (w_0 + w_1 x_1)) x_1 \\ &= 1 \times 1.5 \\ &= 1.5 \\ error_2^1 &= 10.5 \\ error_3^1 &= 6 \\ error_4^1 &= 15 \\ error_5^1 &= 5 \end{aligned}$$

and:

$$\sum_j (y_j - h_w(x_j) x_j = 38$$

we then run the update of $w_1$

$$\begin{aligned} w_1 &\leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j))x_j \\ w_1 &\leftarrow 0 + 0.01 \times 38 \\ w_1 &= 0.38 \end{aligned}$$

For the second iteration, we have:

$$\begin{aligned} error_1^0 &= 1 - (w_0 + w_1 x_1) \\ &= 1 - (0.115 + 0.38 \times 1.5) \\ &= 0.315 \\ error_2^0 &= 1.555 \\ error_3^0 &= 0.745 \\ error_4^0 &= 0.985 \\ error_5^0 &= 1.625 \end{aligned}$$

so:

$$\sum_j (y_j - h_w(x_j) = 5.225$$

we then run the update of $w_0$:

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j)) \\ w_0 &\leftarrow 0.115 + 0.01 \times 5.225 \\ w_0 &= 0.167 \end{aligned}$$

and:

$$\begin{aligned} error_1^1 &= (1 - (w_0 + w_1 x_1))x_1 \\ &= (1 - (0.115 + 1.5 \times 0.38)1.5 \\ &= 0.473 \\ error_2^1 &= 5.44 \\ error_3^1 &= 2.24 \\ error_4^1 &= 4.93 \\ error_5^1 &= 3.25 \end{aligned}$$

and:

$$\sum_j (y_j - h_w(x_j)x_j = 16.33$$

we then run the update of $w_1$

$$\begin{aligned} w_1 &\leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j))x_j \\ w_1 &\leftarrow 0.38 + 0.01 \times 16.33 \\ w_1 &= 0.543 \end{aligned}$$

(Note that these numbers are rounded)

So you can see the way that the gradient descent is working. The error prompts changes in the weights, and these reduce the error.

The way these values change is sensitive to the value of $\alpha$ — with larger values, such as the 0.2 which I has originally posted as the step size to use, the update overshoots the weights.

(b) For stochastic gradient descent, we do the updates after each examples is considered. So:

- $w_0 = 0,\ w_1 = 0,\ x = 1.5,\ y = 1$

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha(y - (w_0 + w_1 x) \\ &\leftarrow 0 + 0.01 \times (1 - (0 + 0 \times 1.5)) \\ &= 0.01 \\ w_1 &\leftarrow w_1 + \alpha(y - (w_0 + w_1 x))x \\ &\leftarrow 0 + 0.01 \times (1 - (0 + 0 \times 1.5)) \times 1.5 \\ &= 0.015 \end{aligned}$$

- $w_0 = 0.01,\ w_1 = 0.015,\ x = 3.5,\ y = 3$

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha(y - (w_0 + w_1 x) \\ &\leftarrow 0.01 + 0.01 \times (3 - (0.01 + 0.015 \times 3.5)) \\ &= 0.0394 \\ w_1 &\leftarrow w_1 + \alpha(y - (w_0 + w_1 x))x \\ &\leftarrow 0.015 + 0.01 \times (3 - (0.01 + 0.015 \times 3.5) \times 3.5 \\ &= 0.118 \end{aligned}$$

- $w_0 = 0.0394,\ w_1 = 0.0.118,\ x = 3,\ y = 2$

$$\begin{aligned} w_0 &= 0.0554 \\ w_1 &= 0.166 \end{aligned}$$

- $w_0 = 0.0554,\ w_1 = 0.166,\ x = 5,\ y = 3$

$$\begin{aligned} w_0 &= 0.0766 \\ w_1 &= 0.272 \end{aligned}$$

- $w_0 = 0.0766,\ w_1 = 0.272,\ x = 2,\ y = 2.5$

$$\begin{aligned} w_0 &= 0.0956 \\ w_1 &= 0.309 \end{aligned}$$

Again, note the rounding of the intermediate calculations.

In this case, where all the values are pretty well grouped around the "true" line, and the step size is small, stochastic gradient descent makes steady progress towards finding the values.

Pushing $\alpha$ up just a little (to 0.03) results in oscillation — values increase and then decrease.

Compared with batch gradient descent, in this case stochastic gradient descent makes slower progress, because the update is being applied to smaller numbers (individual errors not sums). However this is very much an artifact of the training set. Noisier data can make the values generated by stochastic gradient descent jump around much more than those generated by batch, which tends to allow positive and negative errors to balance each other.

A useful exercise is to program this example into a spreadsheet so you can play with different learning rates.

5. (a) This is a version of the equation on page 60.

   We have variables $x_1$ and $x_2$ with weights $w_1$ and $w_2$. We could then update as follows (for stochastic gradient descent):

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha(y - (w_0 + w_1x_1 + w_2x_2) \\ w_1 &\leftarrow w_1 + \alpha(y - (w_0 + w_1x_1 + w_2x_2))x_1 \\ w_2 &\leftarrow w_2 + \alpha(y - (w_0 + w_1x_1 + w_2x_2))x_2 \end{aligned}$$

   When implementing this is would be convenient if all three updates were doing the same basic computation because then we could implement them all as one function, so we rewrite the first of these update rules to get:

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha(y - (w_0 + w_1x_1 + w_2x_2))x_0 \\ w_1 &\leftarrow w_1 + \alpha(y - (w_0 + w_1x_1 + w_2x_2))x_1 \\ w_2 &\leftarrow w_2 + \alpha(y - (w_0 + w_1x_1 + w_2x_2))x_2 \end{aligned}$$

   where $x_0 = 1$.

   (b) As with the single variable method in the previous question, the result is very dependent on the learning rate. A larger learning rate will move the weights towards the "true" values (not that we know what true is) more quickly, but runs the risk of stepping past the true values. When this happens, the adjustment will reverse direction, but can keep overstepping as it oscillates around the right values.

## Version list

- Version 1.0, January 18th 2020.
- Version 1.1, January 11th 2021.
- Version 1.2, January 15th 2022.