COMP3411/9414 Artificial Intelligence
Session 1, 2019

Tutorial Solutions - Week 8

This page was last updated: 04/30/2019 06:06:55

Activity 8.1 Decision Trees

Consider the task of predicting whether children are likely to be hired to play members of the Von Trapp Family in a production of The Sound of Music, based on these data:

height	hair	eyes	hired
short	blond	blue	+
tall	red	blue	+
tall	blond	blue	+
tall	blond	brown	–
short	dark	blue	–
tall	dark	blue	–
tall	dark	brown	–
short	blond	brown	–

a. Compute the information (entropy) gain for each of the three attributes (height, hair, eyes) in terms of classifying objects as belonging to the class, + or – .

There are 3 objects in class '+' and 5 in '-', so the entropy is:

Entropy(parent) = Σ_i P_i $\log_2 P_i$ = -(3/8)log(3/8) - (5/8)log(5/8) = 0.954

Suppose we split on height:

Of the 3 'short' items, 1 is '+' and 2 are '-', so Entropy(short) = -(1/3)log(1/3) - (2/3)log(2/3) = 0.918

Of the 5 'tall' items, 2 are '+' and 3 are '-', so Entropy(tall) = -(2/5)log(2/5) - (3/5)log(3/5) = 0.971

The average entropy after splitting on 'height' is Entropy(height) = (3/8)(0.918) + (5/8)(0.971) = 0.951

The information gained by testing this attribute is: 0.954 - 0.951 = 0.003 (i.e. very little)

If we try splitting on 'hair' we find that the branch for 'dark' has 3 items, all '-' and the branch for 'red' has 1 item, in '+'. Thus, these branches require no further information to make a decision. The branch for 'blond' has 2 '+' and 2 '-' items and so requires 1 bit. That is,

Entropy(hair) = (3/8)(0) + (1/8)(0) + (4/8)(1) = 0.5

and the information gained by testing hair is 0.954 - 0.5 = 0.454 bits.

By a similar calculation, the entropy for testing 'eyes' is (5/8)(0.971) + (3/8)(0) = 0.607, so the information gained is 0.954 - 0.607 = 0.347 bits.

Thus 'hair' gives us the maximum information gain.

b. Construct a decision tree based on the minimum entropy principle.

Since the 'blond' branch for hair still contains a mixed population, we need to apply the procedure recursively to these four items. Note that we now only need to test 'height' and 'eyes' since the 'hair' attribute has already been used. If we split on 'height', the branch for 'tall' and 'short' will each contain one '+' and one '-', so the entropy gain is zero. If we split on 'eyes', the 'blue' brach contains two '+'s and the 'brown' branch two '-'s, so the tree is complete:

Activity 8.2 Laplace Pruning

The Laplace error estimate for pruning a node in a Decision Tree is given by:

where N is the total number of items, n is the number of items in the majority class and k is the number of classes. Given the following subtree, should the children be pruned or not? Show your calculations.

Error(Parent) = 1 - (7+1)/(11+2) = 1 - 8/13 = 5/13 = 0.385

Error(Left) = 1 - (2+1)/(3+2) = 1 - 3/5 = 2/5 = 0.4

Error(Right) = 1 - (6+1)/(8+2) = 1 - 7/10 = 3/10 = 0.3

Backed Up Error = (3/11)*(0.4) + (8/11)*(0.3) = 0.327 < 0.385

Since Error of Parent is larger than Backed Up Error ⇒ Don't Prune

---

Activity 8.3 Perceptron Learning

---

1. Construct by hand a Perceptron which correctly classifies the following data; use your knowledge of plane geometry to choose appropriate values for the weights w_0, w_1 and w_2.

Training Example	x_1	x_2	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

The first step is to plot the data on a 2-D graph, and draw a line which separates the positive from the negative data points:

This line has slope -1/2 and x_2-intersect 5/4, so its equation is:

$x_2 = 5/4 - x_1/2$,
i.e. $2x_1 + 4x_2 - 5 = 0$.

Taking account of which side is positive, this corresponds to these weights:

$w_0 = -5$
$w_1 = 2$

$w_2 = 4$

Alternatively, we can derive weights w_1=1 and w_2=2 by drawing a vector normal to the separating line, in the direction pointing towards the positive data points:

The bias weight w_0 can then be found by computing the dot product of the normal vector with a perpendicular vector from the separating line to the origin. In this case $w_0 = 1(-0.5) + 2(-1) = -2.5$

(Note: these weights differ from the previous ones by a normalizing constant, which is fine for a Perceptron)

2. Demonstrate the Perceptron Learning Algorithm on the above data, using a learning rate of 1.0 and initial weight values of

$w_0 = - 0.5$
$w_1 = 0$
$w_2 = 1$

In your answer, you should clearly indicate the new weight values at the end of each training step.

Iteration	w_0	w_1	w_2	Training Example	x_1	x_2	Class	$s=w_0+w_1x_1+w_2x_2$	Action
1	-0.5	0	1	a.	0	1	-	+0.5	Subtract
2	-1.5	0	0	b.	2	0	-	-1.5	None
3	-1.5	0	0	c.	1	1	+	-1.5	Add
4	-0.5	1	1	a.	0	1	-	+0.5	Subtract
5	-1.5	1	0	b.	2	0	-	+0.5	Subtract
6	-2.5	-1	0	c.	1	1	+	-3.5	Add
7	-1.5	0	1	a.	0	1	-	-0.5	None
8	-1.5	0	1	b.	2	0	-	-1.5	None
9	-1.5	0	1	c.	1	1	+	-0.5	Add
10	-0.5	1	2	a.	0	1	-	+1.5	Subtract
11	-1.5	1	1	b.	2	0	-	+0.5	Subtract

12	-2.5	-1	1	c.	1	1	+	-2.5	Add
13	-1.5	0	2	a.	0	1	-	+0.5	Subtract
14	-2.5	0	1	b.	2	0	-	-2.5	None
15	-2.5	0	1	c.	1	1	+	-1.5	Add
16	-1.5	1	2	a.	0	1	-	+0.5	Subtract
17	-2.5	1	1	b.	2	0	-	-0.5	None
18	-2.5	1	1	c.	1	1	+	-0.5	Add
19	-1.5	2	2	a.	0	1	-	+0.5	Subtract
20	-2.5	2	1	b.	2	0	-	+1.5	Subtract
21	-3.5	0	1	c.	1	1	+	-2.5	Add
22	-2.5	1	2	a.	0	1	-	-0.5	None
23	-2.5	1	2	b.	2	0	-	-0.5	None
24	-2.5	1	2	c.	1	1	+	+0.5	None

Activity 9.1 Multi-Layer Neural Networks to Compute Logical Functions

---

Explain how each of the following could be constructed:

1. Perceptron to compute the OR function of *m* inputs

   Set the bias weight to -½, all other weights to 1.

   The OR function is almost always True. The only way it can be False is if all inputs are 0. Therefore, we set the bias to be slightly less than zero for this input.

2. Perceptron to compute the AND function of *n* inputs

   Set the bias weight to (½ - *n*), all other weights to 1.

   The AND function is almost always False. The only way it can be True is if all inputs are 1. Therefore, we set the bias so that, when all inputs are 1, the combined sum is slightly greater than 0.

3. 2-Layer Neural Network to compute any (given) logical expression, assuming it is written in Conjunctive Normal Form.

Each hidden node should compute one disjunctive term in the expression. The weights should be -1 for items that are negated, +1 for the others. The bias should be (k - ½) where k is the number of items that are negated. The output node then computes the conjunction of all the hidden nodes, as in part 2.

For example, here is a network that computes (A ∨ B) ∧ (¬ B ∨ C ∨ ¬ D) ∧ (D ∨ ¬ E)