19T2: COMP9417 Machine Learning and Data Mining

Lectures: Tree Learning
Topic: Questions from lectures
Version: with answers
Last revision: Fri Jun 28 09:57:48 AEST 2019

Introduction

Some questions and exercises from the course lectures covering aspects of supervised tree learning for classification and regression.

Expressiveness of decision trees

Question 1 Give decision trees to represent the following Boolean functions, where the variables `A`, `B`, `C` and `D` have values `t` or `f`, and the class value is either `True` or `False`:

a) $A \wedge \neg B$

b) $A \vee [B \wedge C]$

c) $A \;\; X\text{OR} \;\; B$

d) $[A \wedge B] \vee [C \wedge D]$

Can you observe any effect of the increasing complexity of the functions on the form of their expression as decision trees ?

Answer

a) $A \wedge \neg B$:

```
A = t:
|   B = f:  True
|   B = t:  False
A = f:  False
```

b) $A \vee [B \wedge C]$

```
A = t:  True
A = f:
|   B = f:  False
|   B = t:
|   |   C = t:  True
|   |   C = f:  False
```

c) $A\ X\text{OR}\ B$

```
A = t:
|   B = t:  False
|   B = f:  True
A = f:
|   B = t:  True
|   B = f:  False
```

d) $[A \wedge B] \vee [C \wedge D]$

```
A = t:
|   B = t:  True
|   B = f:
|   |   C = t:
|   |   |   D = t:  True
|   |   |   D = f:  False
|   |   C = f:  False
A = f:
|   C = t:
|   |   D = t:  True
|   |   D = f:  False
|   C = f:  False
```

Notice the *replication* effect of repeated subtrees as the target expression becomes more complex, for example, in the tree for *d*. This is a situation where the hypothesis class of models (here, decision trees) can fit any Boolean function, but in order to represent the function the tree may need to be very complex. This makes it hard to learn, and will require a lot data !

Decision tree learning

Question 2 Here is small dataset for a two-class prediction task. There are 4 attributes, and the class is in the rightmost column. Look at the examples. Can you guess which attribute(s) will be most predictive of the class ?

species	**rebel**	**age**	**ability**	**homeworld**
pearl	yes	6000	regeneration	no
bismuth	yes	8000	regeneration	no
pearl	no	6000	weapon-summoning	no
garnet	yes	5000	regeneration	no
amethyst	no	6000	shapeshifting	no
amethyst	yes	5000	shapeshifting	no
garnet	yes	6000	weapon-summoning	no
diamond	no	6000	regeneration	yes
diamond	no	8000	regeneration	yes
amethyst	no	5000	shapeshifting	yes
pearl	no	8000	shapeshifting	yes
jasper	no	6000	weapon-summoning	yes

You probably guessed that attributes 3 and 4 were not very predictive of the class, which is true. However, you might be surprised to learn that attribute “species” has higher information gain than attribute ”rebel”. Why is this ? Refer to slides 71–73 on “Attributes with Many Values” in the lecture notes.

Suppose you are told the following: for attribute “species” the Information Gain is 0.52 and *Split Information* is 2.46, whereas for attribute “rebel” the Information Gain is 0.48 and *Split Information* is 0.98.

Which attribute would the decision-tree learning algorithm select as the split when using the *Gain Ratio* criterion instead of Information Gain ? Is Gain Ratio a better criterion than Information Gain in this case ?

Answer

Gain Ratio corrects for a bias in Information Gain that favours attributes with many values by dividing it by the Split Information, which is the information of the partition of the data according to the values of the attribute in question. This is shown in the formula on slide 72 for Split Information of some attribute A on a sample S, where c is the number of values for A.

For attribute “species” Gain Ratio is $\frac{0.52}{2.46} \simeq 0.21$, whereas for attribute “rebel” Gain Ratio is $\frac{0.48}{0.98} \simeq 0.49$.

So attribute “rebel” would be selected under the Gain Ratio criterion.

In this case, it is a better choice, since the “species” attribute has higher information gain simply by having many more values in this dataset. Note however that other corrections to information gain have been proposed that have other advantages in different settings, and in general there is no “best” splitting criterion for all settings.

Question 3 Assume we learn a decision tree to predict class Y given attributes A, B and C from the following training set, with no pruning.

A	B	C	Y
0	0	0	0
0	0	1	0
0	0	1	0
0	1	0	0
0	1	1	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	0	1
1	1	1	0
1	1	1	1

What would be the training set error for this dataset ? Express your answer as the number of examples out of twelve that would be misclassified.

Answer

2. There are two pairs of examples with the same values for attributes A, B and C but a different (contradictory) value for class Y. One example from each of these pairs will always be misclassified (noise).

Question 4 One nice feature of decision tree learners is that they can learn trees to do *multi-class* classification, i.e., where the problem is to learn to classify each instance into exactly one of $k > 2$ classes.

Suppose a decision tree is to be learned on an arbitrary set of data where each instance has a discrete class value in one of $k > 2$ classes. What is the maximum training set error, expressed as a fraction, that any dataset could have ?

Answer

First consider the case where all k class values are evenly distributed, so there are $\frac{1}{k}$ examples of each class in the training set. In the worst case a decision tree can be learned that predicts one of the k classes for all examples. This will get $\frac{1}{k}$ of the training set correct, and make $k - 1 \times \frac{1}{k}$ mistakes, or $\frac{k-1}{k}$ as a fraction of the training set.

If any one class has more than $\frac{1}{k}$ examples then the worst case decision tree is guaranteed to predict that class, which will reduce the error since that class now represents more than $\frac{1}{k}$ of the training set.

Model trees

Question 5 In learning a model tree, which is a piecewise-linear approximation to a function, it may be necessary to balance predictions made by linear models at the leaves, which possibly are based on small samples of the training set, with the "coarser" linear models fitted at internal nodes. Apply the *smoothing heuristic* shown on slide 93 of the lecture notes on "Tree Learning" for the following situation.

Suppose you have a univariate target function which for the value $x = 2$ is well approximated by $y = 2x + 1$. A model tree has been learned with an internal mode containing the linear model $y = 1.5x + 1$. At this internal node there is a split with condition $x \leq 2$ below which is the leaf node containing the linear model $y = 5x + 1$. Say the number of examples from the training set used to learn the leaf node was 2, and let's say the user has decided to apply a smoothing constant of 5. Compute the true value of y for $x = 2$, the value that will be returned by the leaf node, and the value that will be pased up the tree by the internal node. Has the smoothing improved the prediction ? Comment on the role of n and k as they are used in this formula.

Answer

The true value of the function for $x = 2$ is $y = 2 \times 2 + 1 = 5$.
The leaf node will give $y = 5 \times 2 + 1 = 11$. The internal node will give $y = 1.5 \times 2 + 1 = 4$.
The smoothing function is:

$$p' = \frac{np + kq}{n + k}$$

where p' is the "smoothed" prediction to be passed up the tree, p is the value computed by the linear model at the node below the current node (here, p comes from the leaf node) and n is the number of training examples used to compute the value at that node, q is the value at the current node (here, the node with the split $x \leq 2$), and finally k is the smooting constant (think of it as the number of "virtual" examples that will be used to give weight to the prediction at the current node, compared to the node below).
For the values we have this gives:

$$p' = \frac{(2 \times 11) + (5 \times 4)}{2 + 5} = 6$$

Clearly the smoothed value to be passed up the tree is closer to the actual value than that calculated by the leaf, so smoothing has improved the estimate of y.
The parameters n and k have the role of balancing how much we "believe" in the lower nodes in the tree, which will be learned on smaller amounts of data, compared to their parent nodes.

Extending Tree Learning

Question 6 Propose an algorithm extending decision trees with Naive Bayes classification. Explain what changes to the basic TDIDT learning algorithm: a) during training; and b) at classification time.

Answer

One way to implement this is the following. When learning a tree top-down, at each node, consider splitting the node by each attribute. However, instead of computing the entropy at each new leaf, evaluate the fit of a *Naive Bayes classifier (NBC)* to the sample of data at that leaf. If the weighted sum of each of the leaf nodes' NBC's (cross-validated) accuracies does not improve by more than a given threshold on the accuracy of the parent node, set the parent node to be a leaf and halt. Otherwise, recursively descend into each of the leaves and try splitting further into sub-trees.

This could be a good solution if you have lots of data, since the hybrid tree/NBC can exploit the advantages of both approaches. The tree will split the data into subsets where it is less likely that the NBC's independence assumptions will be violated, and the tree growth may be halted more quickly since the NBC can use all the attributes of the data in each sample to make its predictions. It should not be too inefficient, since when cross-validating the NBC's counts are all that have to be updated.

When testing, an example should just get passed down the tree in the usual way until it hits the correct leaf, where its class will be predicted by the NBC with a probability.

Question 7 How could you combine tree learning with *local regression* as described on slides 101–105 of the week 1 lecture "Regression" ?

Answer

Local regression is a nearest-neighbour technique for predicting a numeric output given a set of labelled training data and a distance function. It could be combined with tree learning in the following way. Start by learning a regression tree, but for each leaf node *save* all the training data appearing at that leaf. Then to predict the class value of any query instance, pass the instance down the tree. When it reaches a leaf node, apply local regression *using the examples associated with that leaf* to predict the value. A refinement of this would be to use only the attributes on the path from root to leaf to compute the distance measure.

Question 8 The space of possible decision trees can be enormous, so searching for the "best" tree to fit a given dataset is usually not doneIn fact, under certain conditions on what the "best" tree is the problem is known to be NP-complete., with the most common solution being the greedy search of the basic TDIDT algorithm. Suggest how you could extend the basic greedy approach to allow a form of backtracking search.

Answer

There are a number of options here, however, simply applying something like a standard depth-first search (DFS) may not be a good solution. This is because if your method is trying to minimise some function of training set error a DFS may end up searching a huge number of trees. A better option would be to bound the number of possible splits considered, say by using a *beam search*[1] where for each node considered when constructing the tree the search maintains an ordered list of the k best possible attributes to split on, where k is the size of the "beam". This is essentially a relaxation of the standard greedy search, and is related to methods where trees are used in ensemble learning.

[1]See `http://en.wikipedia.org/wiki/Beam_search`.