CS 478

Scott Leland Crossen

Perceptron Project

Part 1

The code that implements the ID3 decision tree algorithm is included with the submission of this project. It is written in Scala and follows the functional paradigm. Immutability is observed in all but the underlying learner. Unknown attributes are handled in accordance with the class suggestions as well as the other ways discussed in part 6. To check the algorithm, the lenses data set returned an accuracy between 66% and 100% on a non-pruned 75/25 testing/training split.

Part 2

As per the TAs suggestion, I used method 1 of part 6 described in this report when handling missing values.

Out[25]=

Accuracy of Testing Sets

Fold	Cars Train Accuracy	Cars Test Accuracy	Voting Train Accuracy	Voting Test
01	1	0.947674	1	0.9069
02	1	0.971098	1	0.8636
03	1	0.965318	1	0.9767
04	1	0.919075	1	0.9545
05	1	0.936416	1	0.9069
06	1	0.947674	1	0.9318
07	1	0.913295	1	1.
08	1	0.930636	1	0.9772
09	1	0.965318	1	1.
10	1	0.947977	1	0.9318
Avg	**1**	**0.944448**	**1**	**0.9449**

Which is within the expected range given in the problem. Note that without a stopping criteria the training accuracy will be 100% as long as there are no two identical feature selections that have different labels. If this condition does not exist then there will be at least one non-pure leaf node in the resulting tree. In my opinion it is also likely that the unity accuracies on the 7th and 9th folds for the voting set can be attributed to duplicate features in the testing sets.

Part 3

Cars Set

For the cars data set, the most important features appeared to be “safety” (usually the root node) and “persons” (the next most important node for the next 2/3 of the branches). One interesting behavior was that the tree would form a pure node at the low-safety branch of the root node. Unlike the voting set, this tree would usually define itself to a depth equal to the amount of features. Because of the limited amount of features and the large amount of data, the tree was usually very shallow but very wide. As mentioned the “safety” and “persons” attributes were the features that most distinctly partitioned the data in the best groups. This data set doesn’t seem to have missing values and so did not form any interesting structures because of that.

Voting Set

For every fold in the voting set the root node was “physician-fee-freeze”. Consequently the next nodes down were “synfuels-corporation-cutback” for the “yes” branch and “adoption-of-the-budget-resolution” for the “no” branch. This means that without fail the most import attributes were these three. They were the features that most distinctly partitioned the data in the best groups. Interestingly, there were 16 features in the voting data set but each tree showed a maximum depth of about 6 features. This means that 10 of the included features are redundant. It’s possible that they are all used except in different branches but this doesn’t seem to be the case as there are a few features that are obviously not included just by observation. Another interesting fact is that those rows with missing data are often easily classified with method 1 of part 6 of this report.

Part 4

In order to choose a good method for which to use for handling missing values I decided to speak with a TA. After discussing the problem with him, I came up with a few good ways to handle the situation. I implemented the best three but am basing the first few problems of this report on the first method described. The rest are described in part 6 of this report. The first method involves treating missing values as a separate enumeration of the selected feature by themselves. This would form missing-value nodes that would total the probabilities of the likely results for use in prediction. It’s possible that these nodes won’t be formed for every instance of testing data so in these cases the node would defer evaluation of the attribute row to all of it’s children. The child nodes would recursively return label predictions with associated certainties. The parent node would then dot-product the prediction certainties with the node frequency for each child to get a set of probabilities per label for each child. It would then sum similar-label across all children probabilities and pick the largest as the prediction.

I used this approach mostly because the TA suggested it and I wanted to get full points for the project (just being honest here). However, I did have to use some invention to handle the case of testing data that included missing-nodes never seen in the training set. I found that what I did worked very well. This method reduced the missing-value problem to base probabilities the best and so I used it.

Part 5

For this part I used a 75/25 training/validation split on the training data. The same ratio was also used on the training/testing split of the data.

Pruned/Unpruned Comparison

Parameter	Cars Unpruned	Voting Unpruned	Cars Pruned	Voting Pruned
#Nodes	371	58	255	20
Depth	7	9	7	6
Accuracy	0.944448	0.944979	0.925926	0.968248

As is evident in the table, pruning usually reduces both the depth and number of nodes for the tree. Though it didn't increase the accuracy on the cars data set, it did significantly increase the average accuracy on the voting data set. This is likely because of the nature of the two data sets. The voting set is more "rich" in the sense that it has more features and possible paths to travel through the tree. The cars data was shallow to begin with (because it has very few features) and pruning didn't have as much opportunity to improve performance. This difference can also likely be because of the level of noise in each data set. The cars data set could have less noise and so pruning would show less effect.

Part 6

For this section I implemented three different ways to handle missing values. Note that in order to work well with pruning, the testing method (as opposed to the training method) for each is implemented the same way. This involves comparing the feature corresponding to a given node with the possible children. If the child exists that matches the feature then recurse on that child. If it doesn't then recurse on all children and assign a weight to each child's prediction corresponding to the probability of the child's appearance in the test set. Sum all probabilities that predicted the same label and then pick the label with the highest prediction.

The first method is described in detail in part 4 of this report. Methods 2 and 3 are described below:

Method 2 involves creating a child node for every possible value of the missing feature in the tree when it is encountered. Under this method, a missing value really counts as every possible value and the probability and information gain is calculated as such. I implemented this method so that the total probability could exceed 1 seen as missing-feature rows will be passed to all branches when the data is split.

Method 3 doesn't account for missing values in the calculation of information gain. The feature is still passed to each child to deal with, but the next best node to split on is not based off this information.

Below is the table comparing these methods using the voting data set. Each method implements the normal pruning strategy

Voting Data Missing Strategy Comparison

Parameter	Method 1	Method 2	Method 3
#Nodes	20	77	124
Depth	6	17	17
Accuracy	0.962477	0.98578	0.944954

As expected, the regular method creates the simplest tree. It also results in a very good average. The second method also creates a good tree. With the second method the un-pruned tree is actually very large but the pruning algorithm allows for unneeded branches to be cut off drastically. The goal was to over-define the tree so that pruning would cut it back to a reasonable level but with the best accuracy possible. As expected, the broad definition of this algorithm also gives it a higher accuracy, but this is at the cost of node depth. the final method is all-around poor. It is the largest tree and also the least accurate. It would appear as if over-defining the tree in this method did not allow the pruning algorithm to increase the accuracy very much at all. The third method was essentially a compromise between the first two and so essentially got both the bad behaviors of them as well.