Decision Tree Report

Wesley Ackerman

Lenses Dataset

I used decision tree to learn the lenses dataset. The tree was trained all the way out, with no pruning. Using 10-fold cross validation, I got an average accuracy of 73.3%. The lenses dataset was great to start with because I was able to visualize the entire tree. I found that splitting on "tear production rate" first yielded the most information, as **all** instances with a reduced tear production rate did not have contact lenses.

Cars Dataset

My decision tree implementation got an average of around 94% accuracy in predicting the cars dataset, using 10-fold cross validation. I noticed that accuracy on the training set was always 100%. This is because I was training the tree all the way out. Nodes were continually split until each node was pure. So, each instance in the training set was represented in the tree, and could be traced all the way to its correct label.

Pruning is a case where you might not get 100% accuracy on the training set. In pruning back some of the nodes, you may end up giving its parent a different classification. Then, when the training set instance is tested, an incorrect output class will be chosen. I also think it is possible to get less than 100% accuracy on the training set with continuous values. Since the decision tree doesn't deal with them as easily, I could see a training instance with a continuous value on the border of a bin being classified incorrectly.

Cars Dataset Performance

	Test Set Accuracy	Training Set Accuracy
Fold 1	0.9651	1
Fold 2	0.9133	1
Fold 3	0.9306	1
Fold 4	0.9653	1
Fold 5	0.9422	1
Fold 6	0.9593	1
Fold 7	0.9133	1
Fold 8	0.9017	1
Fold 9	0.9306	1
Fold 10	0.9075	1
Average	0.9329	1

In the cars dataset, I found that the tree split first on the final attribute, which is "safety". This occurred because, according to the model, every instance which had a "low" safety rating also had an "unaccurate" label. Thus, one of the nodes just below the root didn't have to split any further. The

other two nodes on the same level both split on the "persons" attribute. This means that the number of people a car can hold corresponded a great deal with its label. According to the model, almost all instances with a value of "more" for the "persons" feature had an "unaccurate" label. Thus "safety" and "persons" features were the first two to be split on.

Voting Dataset

My implementation averaged around 95% accuracy on predicting the voting dataset, using 10-fold cross validation. Training set accuracy was always either 99.76% or 100%. I found that there were two conflicting instances in the dataset. Their voting records were exactly the same, but they belonged to different parties. So, nothing about their voting pattern was unique, and the model essentially had to guess on their particular label. Thus, it sometimes got one wrong. This is a case where the tree won't always get 100% accuracy on the dataset: when there are instances that are exactly alike, but with different output classes. The algorithm essentially has to guess in those cases, and will sometimes get it wrong.

Voting Dataset Performance

	Test Set Accuracy	Training Set Accuracy
Fold 1	0.9348	0.9976
Fold 2	0.9565	0.9976
Fold 3	0.913	1
Fold 4	0.9783	0.9976
Fold 5	0.913	0.9976
Fold 6	0.9348	1
Fold 7	0.9783	0.9976
Fold 8	0.9783	0.9976
Fold 9	0.9565	0.9976
Fold 10	1	0.9976
Average	0.9543	0.99808

I found that the tree always split on the "physician-fee-freeze" attribute first, which means that it correlates highly with whether a representative is republican or democrat. On the second and third levels, there was a little more variation in what features were split on. It depended on the exact training set it was presented with. I found that the second and third level nodes split most often on "adoption-of-the-budget-resolution" and "synfuels-corporation-cutback". These features must also correlate a lot with the party of any given representative.

To handle unknown attributes in the voting dataset, I picked the most common value in the column, and made that the new attribute. I did so because I felt it was relatively simple, and it wouldn't skew the data much. I thought about taking an average of all values in the column, but since all attributes were nominal, that didn't make much sense. If I had more time, I would probably try the approach of giving an unknown the most common value in its column, based on the instance's output class. I feel that would be more accurate and true to the data, but I chose not to do it because of time.

Voting Dataset: Pruning

I found that I was able to do a great deal of pruning before accuracy on the validation set began to decline. The fewest number of nodes I was able to get, with the best accuracy, was 19 nodes and 94.5%. I found that accuracies tended to stay in the same ballpark when a node was removed, and only deteriorated rapidly when the final few nodes were removed.

Depth (# Nodes)	Node Count	Accuracy on VS
9	51	0.9189
9	49	0.9324
9	47	0.9324
9	45	0.9324
9	43	0.9189
9	41	0.9459
9	39	0.9459
9	37	0.9459
9	35	0.9459
9	33	0.9459
9	31	0.9459
9	29	0.9459
9	27	0.9459
9	25	0.9459
9	23	0.9459
9	21	0.9324
9	19	0.9459
9	17	0.9324
8	15	0.9324
7	13	0.9324
6	11	0.9324
5	9	0.9189
4	7	0.9189
1	1	0.5676

As these data show, I went about by deleting all of the parent's child nodes, and then going back up and deleting the parent. As I think about it, it probably would have made more sense to prune the nodes row-by-row, since the leaf nodes are often the ones with the most extrapolation. It is clear that accuracy drops to around 50% with only one node, which is equivalent to random guessing in this dataset.

Experimentation

I used gain ratio as the attribute to maximize and split on, rather than information gain. I understand it to mean that the ratio compares the number of feature output values and spread between values again the information that could be gained by splitting on the attribute. In my implementation of information gain, I found that the decision tree performed slightly worse at predicted an instance from the test set. As expected, it performed just as well at predicting the training set. Compare these data to the voting dataset table above:

Voting Dataset Performance (Using Gain Ratio)

	Test Set Accuracy	Training Set Accuracy
Fold 1	0.8913	1
Fold 2	0.8696	0.9976
Fold 3	0.8913	0.9976
Fold 4	0.913	1
Fold 5	0.913	0.9976
Fold 6	0.9348	0.9976
Fold 7	1	1
Fold 8	0.913	0.9976
Fold 9	0.9783	0.9976
Fold 10	0.8511	0.9976
Average	0.9155	0.99832

The model's prediction accuracy was 91.55%, which is worse than its average 95.43% when using information gain as the criteria. It is quite possible that I mis-implemented some small piece of calculating the gain ratio. However, the slides mention that gain ratio is maximized for trivial partitions. It is possible that this would cause the algorithm to choose less-optimal feature splits.