EECS 440
Prof. Soumya Ray

Programming Problem Set 1
9/28/2018
Aaron Weinberg (aaw66), Jeff Eben (jee50), Tristan Maidment (tdm47)

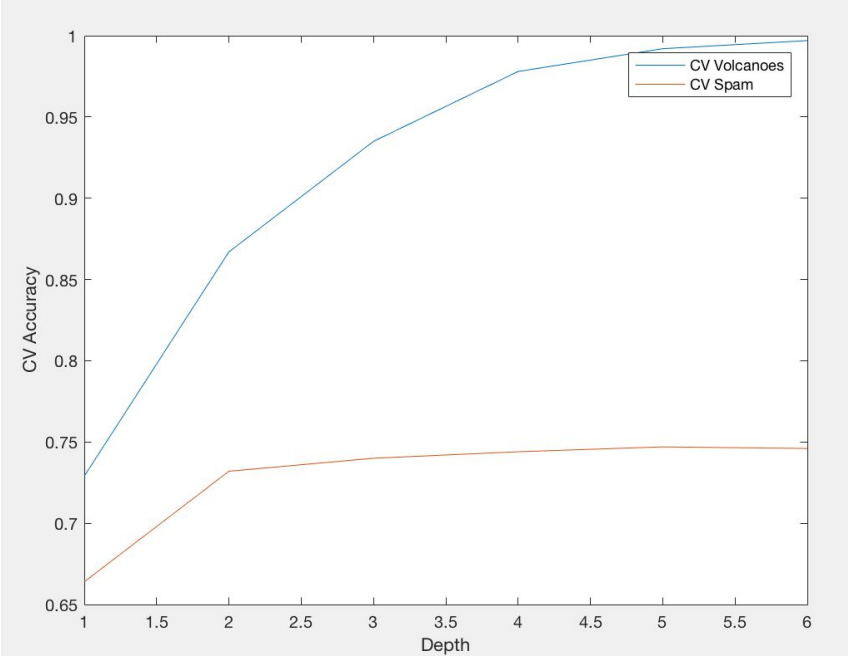
(a) What is the CV accuracy of the classifier on each dataset when the depth is set to 1?

For the Voting dataset, the 5-Fold CV accuracies were [0.966, 1.00, 1.00, 0.977, 1.00], averaging to 0.989 +/- 0.016. For Volcanoes, the 5-Fold CV accuracies were [0.740, 0.720, 0.726, 0.726, 0.735], averaging to 0.729 +/- 0.008. For Spam, the 5-Fold CV accuracies were [0.667, 0.664, 0.662, 0.663, 0.664], averaging to 0.664 +/- 0.002.

(b) For spam and voting, look at first test picked by your tree. Do you think this looks like a sensible test to perform for these problems? Explain.

For the spam dataset, the first tested attribute picked by the tree on all 5 CV iterations was OS. This makes sense since spam emails are most likely going to be coming from servers set up by companies for spamming customers. Therefore, there is probably a high correlation between OS and whether or not the email is spam. For the voting dataset, the first feature selected was "Repealing-the-Job-Killing-Health-Care-Law-Act". This would make sense as a feature that would highly determine which party the voters are for since universal health care is a very controversial topic with a large division between Democrats and Republicans.

(c) For volcanoes and spam, plot the CV accuracy as the depth of the tree is increased. On the x-axis, choose depth values to test so there are at least five evenly spaced points. Does the accuracy improve smoothly as the depth of the tree increases? Can you explain the pattern of the graph?



The graph implies that the accuracy does improve smoothly as the depth of the tree increases. For the volcano dataset, the accuracy increases at an decreasing rate, slowly converging to some value. This makes sense because a depth of 5 for large datasets such as this would greatly increase the accuracy. However, there is a chance of overfitting if the depth keeps increasing which will drop the accuracy. This can be seen in the spam dataset where it starts to decrease in accuracy from point 4 to 5. This is an example of overfitting that is present with our random seed.

(d) Pick 3 different depth values. How do the CV accuracies change for gain and gain ratio for the different problems for these values?

	Gain, D=1	Gain, D=2	Gain, D=3	Gain Ratio, D=1	Gain Ratio, D=2	Gain Ratio, D=3
Voting	0.989	0.993	0.993	0.989	0.993	0.993
Volcanoes	0.729	0.867	0.935	0.672	0.680	0.680
Spam	0.664	0.732	0.740	0.640	0.640	0.720

For voting, the gain and gain ratio is the exact same at all depths, which is most likely due to the fact that the accuracy is high regardless of depth or split criterion due to the simplicity in the dataset. For volcanoes, the gain ratio performed much poorer than the information gain at all depths, which means that it is probably not a good idea to bias attributes with many categories. This may have a correlation with the fact that the volcano data has so many attributes. For spam, the information gain performed better

than the gain ratio at a depth of 1 and 2, but was more similar at a depth of 3. Although the information gain was still higher at depth 3, it was much more similar to gain ratio, meaning that at a higher depth, gain ratio might outperform information gain for spam.

(e) Compare the CV accuracies and the accuracy on the full sample for depths 1 and 2. Are they comparable?

	CV, D=1	CV, D=2	Full, D=1	Full, D=2
Voting	0.989	0.993	0.989	0.993
Volcanoes	0.729	0.867	0.730	0.866
Spam	0.664	0.732	.664	0.732

As seen above, all of the samples improve from a depth of 1 to a depth of 2. When comparing CV to training/testing on the whole sample, all samples have almost the exact same accuracy at both depths. This is expected since the main factor reducing accuracy is the lack of depth, meaning the model is going to underfit the data. Therefore, the model is not able to overfit the training data, which would cause the full sample accuracy to be better than the CV accuracy. This would be seen if we increased depth to a much larger number.

Further Insights/Observations

Working on building the decision tree initially seemed like a straightforward task, but many problems arose when trying to optimize the memory usage, code cleanliness, and interactions with the entropy calculation module. Although the overall structure of the tree was finalized very early in the project, many changes were made in the actual building process of the tree. The structure of the tree is quite simple. The nodes of the tree all extend AbstractNode. There are specific nodes for all variable types, and class labels. Each node contains a dictionary, where the keys are the specific values the attribute can take, and the elements are represent the next node to be conditioned upon.

To build the tree, a recursive function examines nodes that have not had their children expanded. These nodes are stored in a queue in the tree. When considering a node to be expanded, the path from the root is determined, and that path is used to filter the dataset. The maximum entropy of that filtered set is determined, and the child node (dictionary entry) is created. Necessary stopping conditions are set for maxdepth.

Another roadblock was in representation of continuous variables. The entropy calculations provides a split index, but there are many times that a decision tree will have two nodes that indicate a certain range. To account for this range, a special case had to be defined to combine two continuous splits (greater than and less than) into a range. This range is then used for filtering.

When working with the parser provided to calculate the entropy equations, we found that the runtime for the continuous case was far too great to complete building trees for the larger datasets in a

reasonable amount of time. To combat this, we found that by converting the dataset to a pandas DataFrame, with the attribute types as column keys, we were able to greatly increase performance.

Another interesting caveat that we found was in the design of the continuous entropy calculation. Since some datasets, such as the spam dataset, had many continuous attributes with over 75,000 rows of data per attribute, testing the entropy of all possible split values in the dataset initially was a bottleneck. In order to find the split values, we would find potential indices where the dataset might need to be split by looking for when a class went from 0 to 1 or 1 to 0. Then, we would make sure that the splits were not between two attributes with the same value by moving the split index down until it was between two different attributes. However, many edge cases, such as the homogenous group being at the front of the dataset, or the split index becoming a duplicate after being moved, caused the runtime of this algorithm to be extremely long. To combat that, we came up with an algorithm that would first pick viable split indices based on the separation of homogeneous attribute groups. Then, each pair of adjacent attribute groups would be compared to see if their class values were not all the same. This implementation ended up making the logic much less confusing, and also saved a large amount of runtime.