Nick Stevens
nps35

EECS 440 Programming Assignment 1

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

Voting: 0.9841
Volcanoes: 0.6723
Spam: 0.3746

(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.

Voting: Repealing-the-Job-Killing-Health-Care-Law-Act

The examples in this dataset are voting records. Based on this, it would make sense if the classifier had to do with determining the political party of a voter, though I am not certain that this is the case.

Intuitively, this first test makes sense because health care legislation (at least in the United States) tends to be polarizing and indicative of a voter's party affiliation. Furthermore, the wording of the act's title seems like it would provoke extreme reactions, which would translate to a more easily separable and less noisy dataset. This, in turn, would result in a higher information gain value.

Spam: fngr_wss(K)

The examples in this dataset seem to represent emails, which are meant to be classified as "spam" or "not spam."

The most common first-split feature amongst the five folds was fngr_wss(K). I think this is the window size of the packets being sent. This does not seem like it would be a very good indicator of whether an email is spam.

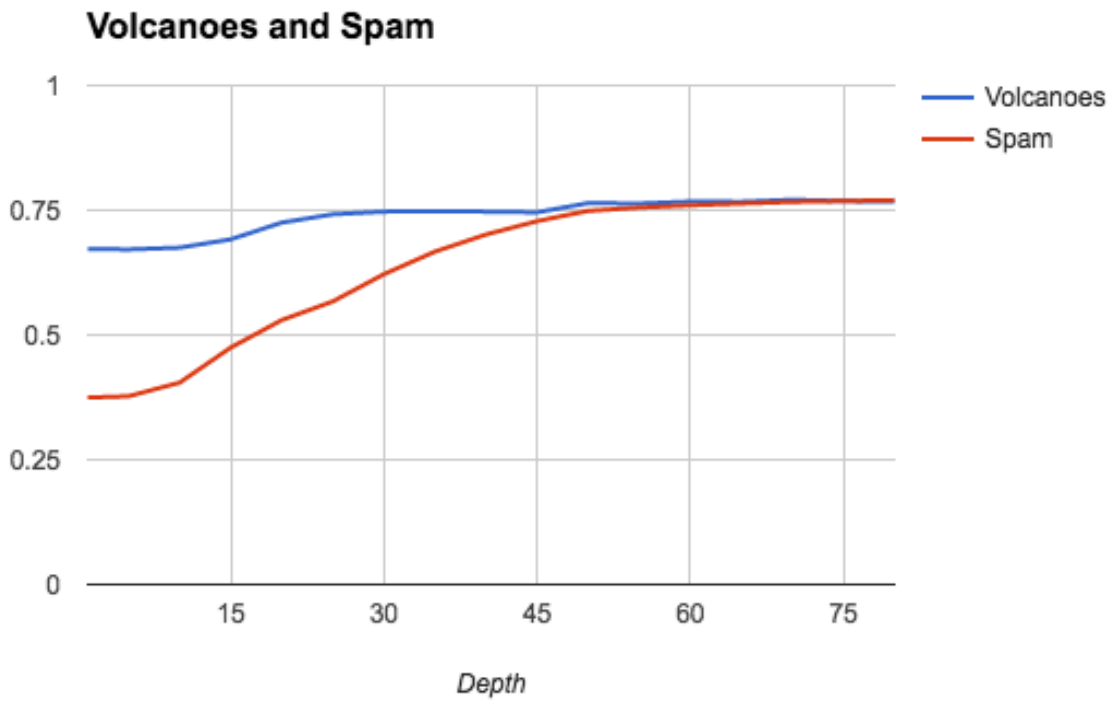
However, for one of the folds, the first-split feature was geoDistance. This seems like a much better indicator of spam because the majority of valid emails are (relatively) local. Emails from halfway around the world are probably more likely to be spam.

Nick Stevens
nps35

(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?

	Average Accuracy	
Depth	**Volcanoes**	**Spam**
1	0.6723	0.3746
5	0.6714	0.3772
10	0.6750	0.4045
15	0.6916	0.4747
20	0.7252	0.5299
25	0.7418	0.5672
30	0.7472	0.6217
35	0.7481	0.6670
40	0.7472	0.7013

	Average Accuracy	
Depth	**Volcanoes**	**Spam**
45	0.7458	0.7280
50	0.7642	0.7487
55	0.7633	0.7553
60	0.7674	0.7599
65	0.7669	0.7634
70	0.7710	0.7667
75	0.7683	0.7685
80	0.7674	0.7703



Nick Stevens
nps35

The accuracy of the classifier does not improve smoothly with the depth of the tree. Below a certain threshold, the depth seems to have little effect on the accuracy. For instance, the volcanoes and spam datasets improve very little between depth 1 and depth 10. Above this threshold, there is a range where increasing the depth has a much greater effect on the accuracy. For the volcanoes dataset, the accuracy increases most quickly over the interval [10, 25]. For the spam dataset, this interval is [10, 50]. Beyond these endpoints, accuracy increases at a much slower rate. At a depth of 80, the accuracy of the volcanoes classifier actually falls by about .01, indicating that the classifier may be beginning to overfit the data in each fold.

In any case, the plateau for both classifiers shows that after a certain depth, the accuracy depends less on the ability to further split a node. Part of the reason is that as the depth increases, the number of nodes for which num_examples <= MIN_EXAMPLES increases. These nodes cannot be split any further even if they have not reached the depth limit, so increasing this depth affects fewer nodes as the tree grows. This explains the diminishing returns in accuracy gain as depth increases.

As a side note: for both datasets, the classifier produced without a depth limit is marginally more accurate than the best data points here. These just take a very long time to execute. One fold of the volcanoes dataset had an accuracy of 0.7763 and a depth of 142. One fold of the spam dataset had an accuracy of 0.7752 and a depth of 248. So unless there is some depth between 80 and these maximum values with a much higher accuracy, overfitting does not seem to be much of a problem in my implementation.

Nick Stevens
nps35

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

Depth	Voting IG	Voting GR	Volcanoes IG	Volcanoes GR	Spam IG	Spam GR
1	0.9841	0.9841	0.6723	0.6723	0.3746	0.3746
10	0.9773	0.9773	0.7248	0.6750	0.5247	0.4045
50	0.9773	0.9773	0.7248	0.7642	0.7838	0.7487

Using Information Gain, the most common feature for Volcanoes was always image_id and the most common feature for Spam was always OS. This is because these are the only nominal features in those datasets, and therefore the only features with more than two values.

For some reason, in the datasets with continuous attributes the information gain values converged much faster than the gain ratio values. The only explanation I can think of for this would be if the nominal attributes also happened to be better predictors of the class label. But I think it is more likely that there is an error in my code preventing my gain ratio calculation from working exactly as it should.

Besides faster convergence, information gain seems to perform worse on higher-depth trees for the volcanoes dataset, but better on higher-depth trees for the spam dataset.

Gain ratio does not affect classifier accuracy on the voting dataset because all of the features are nominal to begin with and take the same set of values.

Nick Stevens
nps35

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

Voting

Depth	CV Accuracy	Full Sample Accuracy
1	0.9841	0.9841
2	0.9841	0.9841

Volcanoes

Depth	CV Accuracy	Full Sample Accuracy
1	0.6723	0.6723
2	0.6719	0.6759

Spam

Depth	CV Accuracy	Full Sample Accuracy
1	0.3746	0.3746
2	0.3746	0.3746

On the full sample, the CV accuracy varies very little between a depth of 1 and a depth of 2. The only thing it affected was the volcanoes dataset. Increasing the depth from 1 to 2 slightly decreased the cross-validated accuracy, but slightly increased the full sample accuracy. Regardless, for all three datasets these accuracies are fairly comparable.

Nick Stevens
nps35

Additional Notes

For a while, I was having a lot of trouble running my program on the spam dataset because of its size. For some of the features, so many potential split values were found that evaluating them took an incredibly long time. I tried several approaches to fix this.

The first was optimizing my code to use as many underlying C methods as possible. This did not help much. I also tried creating heuristics to diminish the number of splits found. One involved only calculating the gain of splits with multiple "same" elements before and after the split (e.g. TTFF, but not TFT), but I think this idea was fundamentally misguided. I also tried only calculating the gain for splits which had a proportionally more-polarized distribution than that of the previous best split found, but this missed a lot of good split values. Eventually, I tried looking at bottlenecks in my code using cProfile. I discovered two things:

1. A lot of my computation time came from evaluating the information gain of each potential split
2. A lot of the computation time within these split evaluations came from inefficiency in the information gain calculation, especially when creating the subsets of examples on either side of the split.

To solve problem 1:

I stopped evaluating the information gain of every possible split point. Instead, I added each of these to a list. Then, I calculated a step interval, k, by taking the log (base 2) of the length of the list. Using this interval, I built a new list of split candidates containing every k^{th} element from the original list. The information gain was only calculated for the elements of this candidate list. Doing this ensures that the number of information gain calculations will only grow logarithmically with the number of splits found, greatly increasing the execution speed on large datasets. Furthermore, it ensures that the split values tested will be evenly distributed, so at the very least a good approximation of the true best split value will be found.

To solve problem 2:

My first step was to create a new new information gain method specific to split evaluations. Since only the above-split and below-split subsets need to be evaluated, the looping behavior of the normal gain calculation was replaced with simple calculations for each condition.

But the majority of the inefficiency came from computing the distribution of class labels for each subset every time the method was called. Eventually, I realized that I only need to look at the distribution of class labels for the examples between consecutive split candidates. This is because the examples in

this range are the only ones that change the subset distributions for a particular split. So my solution was as follows:

a. Take in the previous distributions as parameters
b. Calculate the label distribution of *only* the examples that had switched from the above-split subset to the below-split subset as a result of the new split value
c. For each label value in this "difference" distribution:
    i. Get the number of examples with that value
    ii. Add this number to the previous below-split distribution
    iii. Subtract this number from the previous above-split distribution
d. The resulting distribution is the proper label distribution for the new split value. Using this distribution, calculate the information gain of the split.

This change resulted in a huge speed increase, which made running my code on large datasets feasible. Even so, running the code on the "spam" set with no depth limit takes quite a while.

Other Notes

- As well as a depth limit, I implemented a limit for the minimum number of examples that must be present for the tree to split on a certain feature. This value is the log (base 2) of the length of the example set. This helps control overfitting by ensuring that the tree has a proportionally significant amount of data to draw from when splitting.
- In order to reduce the amount of repeated code and nesting, I calculate the intrinsic value (for gain ratio) alongside the information gain in the same loop. This all takes place in a single "gain" method. If the gain ratio option is not selected, the calculated intrinsic value is just ignored. Calculating both terms in parallel and ignoring one is computationally cheap compared to looping over the feature values twice for each gain ratio computation.
- A lot of my results seem unintuitive, so I think there might be a bug somewhere in my evaluation methods. Especially since using information gain instead of gain ratio gives better overall performance on the Spam dataset. And the results from the Voting dataset change very little with the depth limit, though this could be because it is smaller. I can't pinpoint the problem here, so any feedback here would be appreciated.