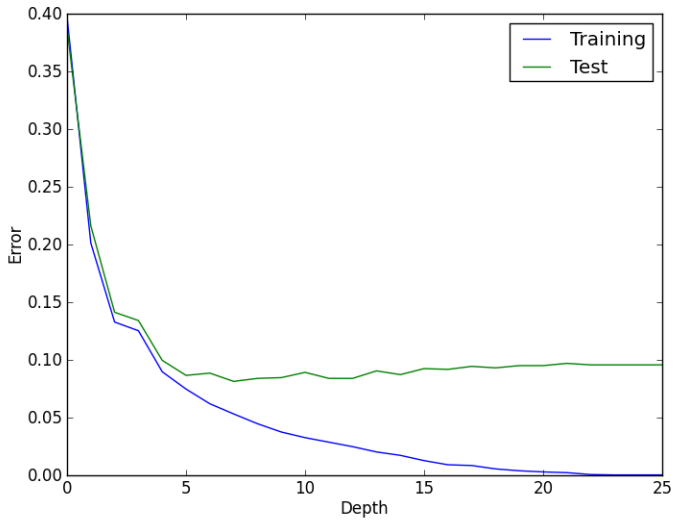
CS189–Spring 2013 — Homework 5

Sharad Vikram, Gerald Fong, Kevin Yanofsky, section ,

April 18, 2013

Decision Tree (with no pruning)

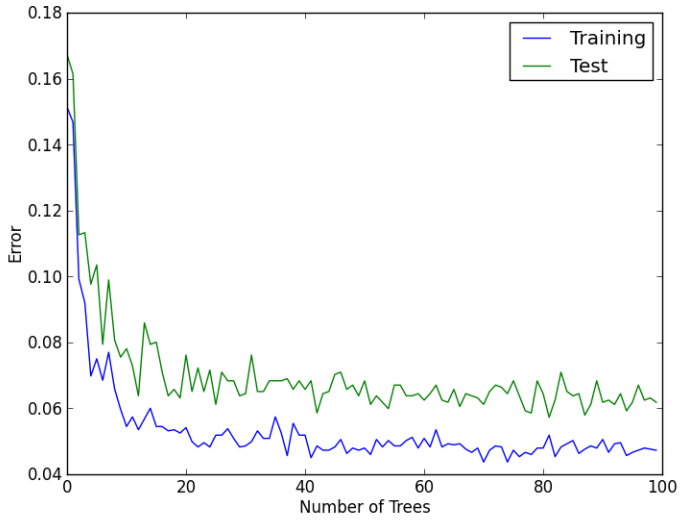


Depth	0	1	2	3	4	5	6	7	8
Training Error	0.397	0.201	0.133	0.125	0.0897	0.0747	0.062	0.0531	0.0447
Test Error	0.387	0.216	0.141	0.134	0.0996	0.0866	0.0885	0.08139	0.084

Best error rate: $e = 0.0813$ with depth 7

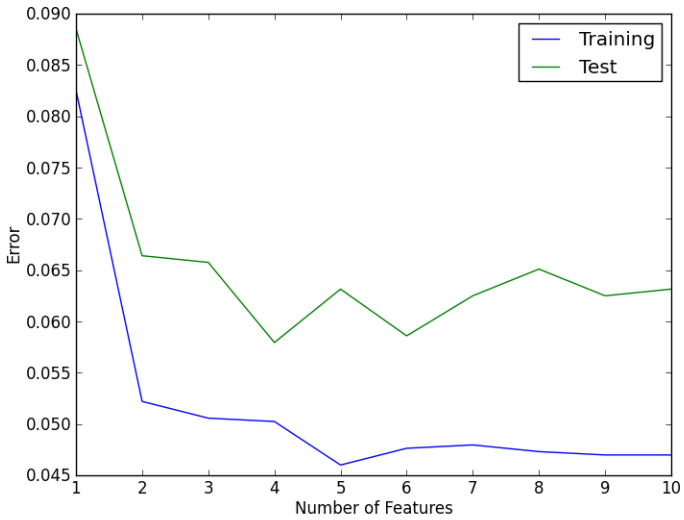
The decision tree has only one parameter which is the maximum depth that a leaf can be at. Once the recursive algorithm reaches a given depth, it will immediately stop splitting and will use a leaf node instead.

Random Forests



7 features, 400 training points, 3 depth

Number of Trees	1	10	20	30	40	50	60	70	80	90	100
Training Error	0.15	0.06	0.05	0.048	0.052	0.047	0.048	0.048	0.048	0.048	0.047
Test Error	0.167	0.076	0.063	0.064	0.068	0.064	0.064	0.063	0.068	0.068	0.062



100 trees, 400 training points, 3 depth

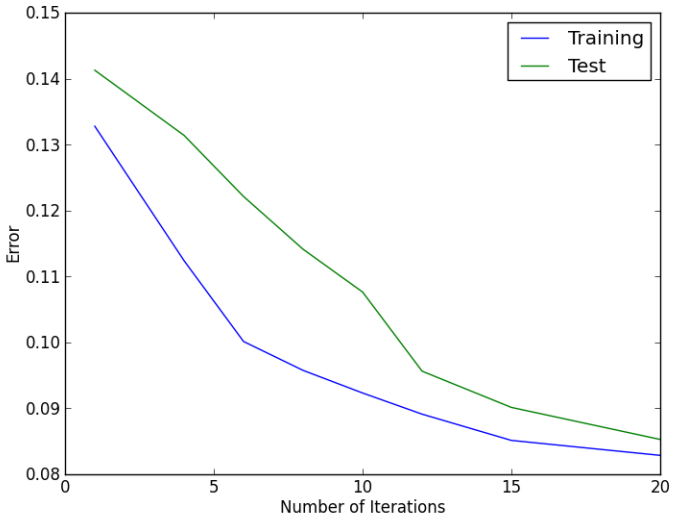
Number of Features	1	2	3	4	5	6	7	8	9	10
Training Error	0.083	0.052	0.051	0.05	0.046	0.048	0.048	0.047	0.047	0.047
Test Error	0.089	0.066	0.066	0.058	0.063	0.059	0.063	0.065	0.063	0.063

Best error rate: $e = 0.059$ with depth 5, 200 points, 100 trees, 6 features

The randomized forest classifier had 4 parameters: maximum tree depth, number of features to split by, number of subsamples of training set to train on, and number of trees.

Tweaking with the parameters led to the result that a low training set number was never beneficial, whereas a lower number of features helped to prevent overfitting and thus improved the error rate. Another trend was that after a reaching a certain number of trees, the error rate would not improve. Similarly, a moderate amount of depth in each tree would also help the error rate.

AdaBoost



Iterations	1	4	6	8	10	12	15	20
Training Error	0.133	0.112	0.1	0.096	0.092	0.089	0.085	0.083
Test Error	0.141	0.131	0.122	0.114	0.108	0.096	0.09	0.085

Best error rate: $e = 0.067$ with 100 iterations

In each iteration, we generated a weak learner (decision tree of depth 5) and trained it using the weighted training set. The learner would split nodes using a weighted entropy measure, and each iteration would train a new learner. The main parameter to be trained was the number of iterations, and a large number of iterations did not hurt.

Introduction to Exploratory Methods

From here on, the implementations were done in matlab in order to have run time gains. The following are more experimental methods and required hire computatinos and custom made datastructures which were better fit in matlab than in python. We tried three different methods with mixed results. First thing we tried was tweaking the way we stopped splitting by having a minimum information gain algorithm. Next, we tried reducing the feature set thresholds so that there was less overfitting. Finally, we tried implementing a two depth greedy algorithm instead of the traditional single depth algorithm.

Decision Tree with Minimum Information Gain Cutoff

Min Information Gain	0.020	0.025	0.030	0.035	0.039	0.080	0.160	0.320
Test Error	0.0768	0.0768	0.0736	0.0710	0.0905	0.0938	0.1536	0.3874

Best error rate: $e = 0.0710$ with $b = 0.39$ iterations

This method was based off of the decision tree algorithm that we implemented in question one. This method was described in the research papers attached with the homework assignment. Essentially, after finding the best information gain that I could get at a particular node after looking through every particular threshold and feature set combination, I would check if the information gain was larging than a threshold beta. If it was, then I would perform the split. Otherwise, I would not perform the split. The advantage of using this algorithm was that we were able to have an unevenly distributed decision tree. If we used purely the depth of the tree as a cutoff point, then if the tree were very imbalanced, then it would result in less effective classification at the larger parts of the tree.

After a lot of tweaking for the beta parameter, we were able to get a fairly decent error rate.

Decision Tree with Reduced Feature Set

Min Information Gain	200	250	300	350	400	450	500	600
Test Error	0.0855	0.0697	0.0697	0.0684	0.0677	0.0716	0.0710	0.071

Best error rate: $e = 0.0677$ with $b = 0.39$ and 400 max thresholds per feature

* All tests used 0.039 minimum information gain using the same algorithm from the last section

One hypothesis we had about we were misclassifying is that we were overfitting the data by looping through every possible question for each of the 57 features. For example,

in total there were 57 features and 13,000 questions (combination of feature and a cutoff value, or "threshold"). In order to reduce this, we implemented a way to set a maximum amount of questions that we would examine for each of the cutoff points. However, we had to pick these points intelligently. We looked at the histogram of points, and picked the number of thresholds that would best represent the data by equally spacing them across each of the data points.

This resulted in a small gain, but was theoretically very cool because we were able to get a smaller error rate with an even smaller amount of computation time with just 1 decision tree!

Decision Tree with N-depth Question Search Algorithm

Best error rate: $e = 0.1034$ with depth 2 search

One thing that we found particular bothering about the initial algorithm of the decision trees is that we make a greedy decision to pick a question which greatest reduces the entropy at that level. This seemed jarring because in the case where you have a 2x2 grid of data with 4 different quadrants, a greedy algorithm would fail to pick the ideal solution by a long shot. Therefore, we came up with the idea of looking ahead one or more levels to decide which tree to pick.

This sounds great in principle, but it turned out to be extremeley difficult to tweak and effectively implement. The computation time of a multiple depth lookahead search algorithm was essentially performing a serach within each search for each of the hundreds of features. We reached a point where computation began too slow even when we introduced one extra depth. We were not able to attain any great results form this, but I believe it is likely due to our slow interation time and inability to correctly perform each of the hyperparameters.

This seems promising if there were a way to reduce the hyperparameters to tweak or have a faster computation time in order to do these rigorous computations.