Handwritten Digit Classification with Bagged Decision Trees

Brian Hall

Introduction

This project was part of an assignment for my Introduction to Machine Learning class. The training set and test set used can be found here, and those files as well as the MATLAB files used for implementation are included in my Github repository.

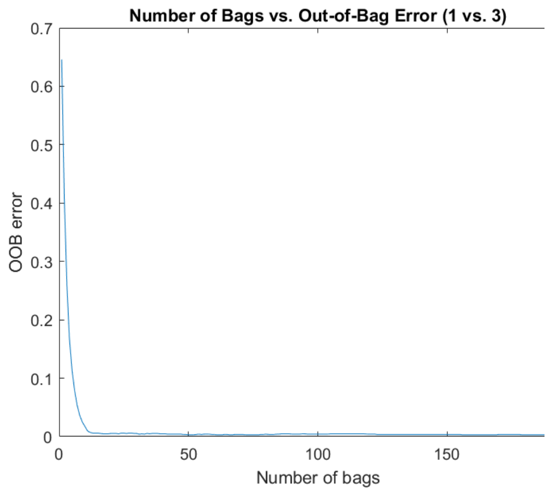
The files zip.train and zip.test contain data representing handwritten 1s, 3s, and 5s. We would like to learn two models, where the first classifies 1s vs 3s (one-vs-three problem) and the second classifies 3s vs 5s (three-vs-five problem). For each problem we filter the data to only include examples of the digits of interest. Our data represents "deslanted and size normalized... 16x16 grayscale images," with 256 features representing the grayscale value of each pixel. The brief description of the data can be found here. Although a convolutional neural network is typically the model of choice for image processing problems, a bagged ensemble should be able to handle binary classification of digits well enough and make predictions faster than a CNN.

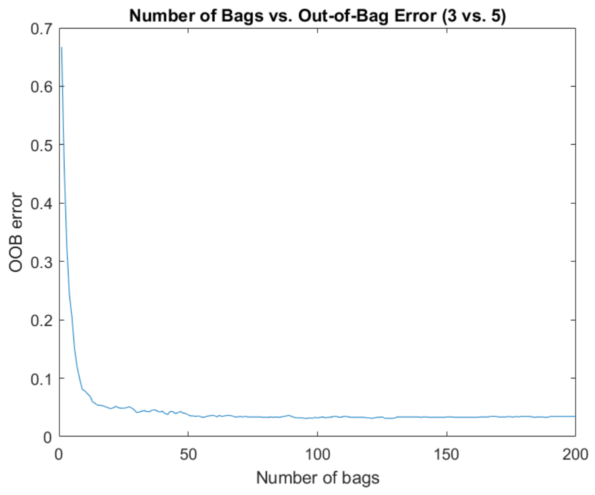
Implementation

Our implementation is focused mainly on bagging, so we won't worry about randomly ignoring features or any other augmentations to the decision trees themselves. Each final model will consist of m max-depth decision trees. Using the training dataset, we create m bootstrapped datasets via random sampling with replacement from the training data. We learn a decision tree for each bootstrapped dataset using MATLAB's `fitctree` function. Then we use each tree to predict a label for the data points that were not selected for that tree's bootstrapped dataset. Finally we use majority voting to aggregate the predictions for each data point in the training set for each model. Our final out-of-bag (OOB) error for our model is the proportion of training examples that are misclassified. As we train new trees we'll keep track of their predictions so we can observe the relationship between OOB error and ensembles of size $m = 1, ..., 200$.

Performance Evaluation

One of the main focuses of this project was to observe the relationship between OOB error and the number of decision trees. The following graphs show number of trees (labelled "bags" in the graph) vs OOB error for the one-vs-three and three-vs-five problems:





In both cases we observe a steep decrease in OOB error up until about 20 to 25 trees, where the error begins to improve much more marginally before converging. This observation gives some nice intuition for the minimum number of trees needed for a reasonably effective model. The figures also seem to suggest that the improved generalization error for a model with a very large number of trees is likely outweighed by the additional time needed to learn those trees. This project was concerned with the case where $m = 200$, which seems like a happy medium between performance and efficiency.

The other focus for this project was comparing the performance of each ensemble of size $m = 200$ to the performance of a single decision tree. We compared the OOB errors with the 10-fold cross-validation error of a single tree, and the test error of each model with the test error of a single tree. These results are summarized in the tables below:

	Cross-validation error of single tree	OOB error of bagged ensemble
One-vs-three	.78%	.24%
Three-vs-five	5.85%	3.38%

In both cases the bagged ensemble outperforms the single tree. We see that the bagged ensemble and the single tree for the three-vs-five problem both perform significantly worse than the ensemble and tree for the one-vs-three problem. Intuitively this makes sense; 1s and 3s are more distinct relative to each other than 3s and 5s, and it's likely that different people draw similar 1s since the shape is so simple (often just a vertical line). For these reasons it's likely that the one-vs-three problem is much easier than the three-vs-five problem, which raises some concerns about the generalization of our three-vs-five model.

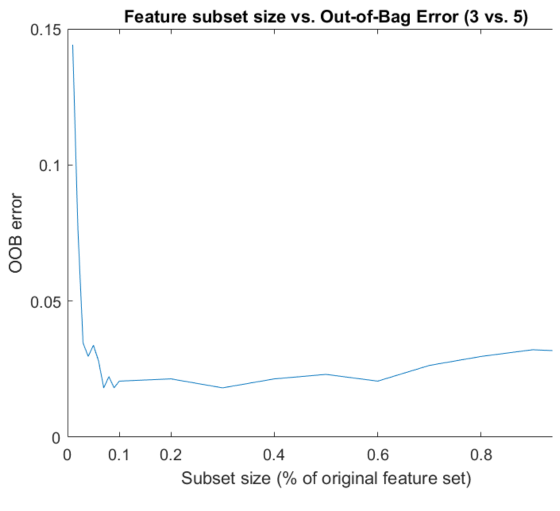
	Test error of single tree	Test error of bagged ensemble
One-vs-three	1.63%	1.16%
Three-vs-five	11.96%	8.90%

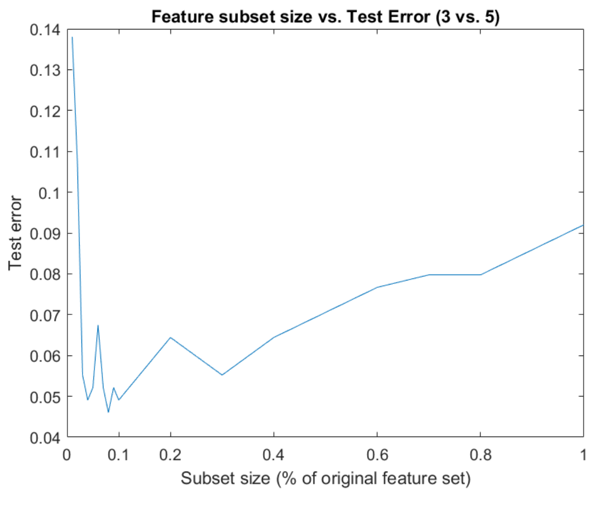
Again the bagged ensemble outperforms the single tree in both cases, but the test error in the three-vs-five case is disappointingly high. Ideally our model would significantly improve upon the performance of a single tree, and these results don't indicate that our model achieves this. Before scrapping the model and methodology entirely for the three-vs-five problem, I'll implement random feature selection in an attempt to improve the model's performance.

Random Feature Selection

In order to leverage the effects of random feature selection, I've modified the original code to allow the user to specify the size of feature subsets to be used to train each tree. Each tree is trained using the same subset size, but the subsets of features themselves are random. I did not remove the bootstrap functionality of the original algorithm, although some sources do advocate for random feature selection without bootstrapping (page 2 of the linked document under "Systematic Creation of Multiple Trees").

The below graphs represent OOB error (left) and test error (right) for 19 models that were trained using different feature subset sizes. The first 10 models were trained using feature subset sizes between 1% and 10% of the original 256 features. The remaining 9 models were trained using feature subset sizes between 20% and 100% of the original 256 features.





In both graphs we see a distinct elbow just below the subset size of 10% of the original 256 features. Beyond this point there is an upward trend in both graphs, which is especially clear for the test error. This is a good indicator that we can improve the generalization of the original model by training individual trees on random subsets of features that are much smaller than the original set of features.

The most likely explanation for this improvement in performance has to do with ensemble learning's reliance on a diverse set of weak learners. As the feature subset size decreases, we learn a more diverse set of trees than from bootstrapping alone, which explains the improvement in OOB error and test error as the size of the feature subset decreases from 100% to ∼10%. But we hit a certain point below 10% where the learners become too weak, and the error rates quickly increase as the subset decreases, resulting in the elbow we see in the graphs.