Problem 1

(a) BaggedTrees.m implementation (results below)
(b) OneThreeFive.m experimentation (results below)

A general trend is that when the number of bags increases, the performance of the overall bagged trees becomes better, most likely because the increasing number of trees decreases the chance of overfitting.

Another observation is that 3-vs-5 problem is harder than 1-vs-3 problem. Either bagged trees or tree trained with cross validation has a larger error measurement in the 3-vs-5 problem. This is because the number three is more similar to the number five than the number one.

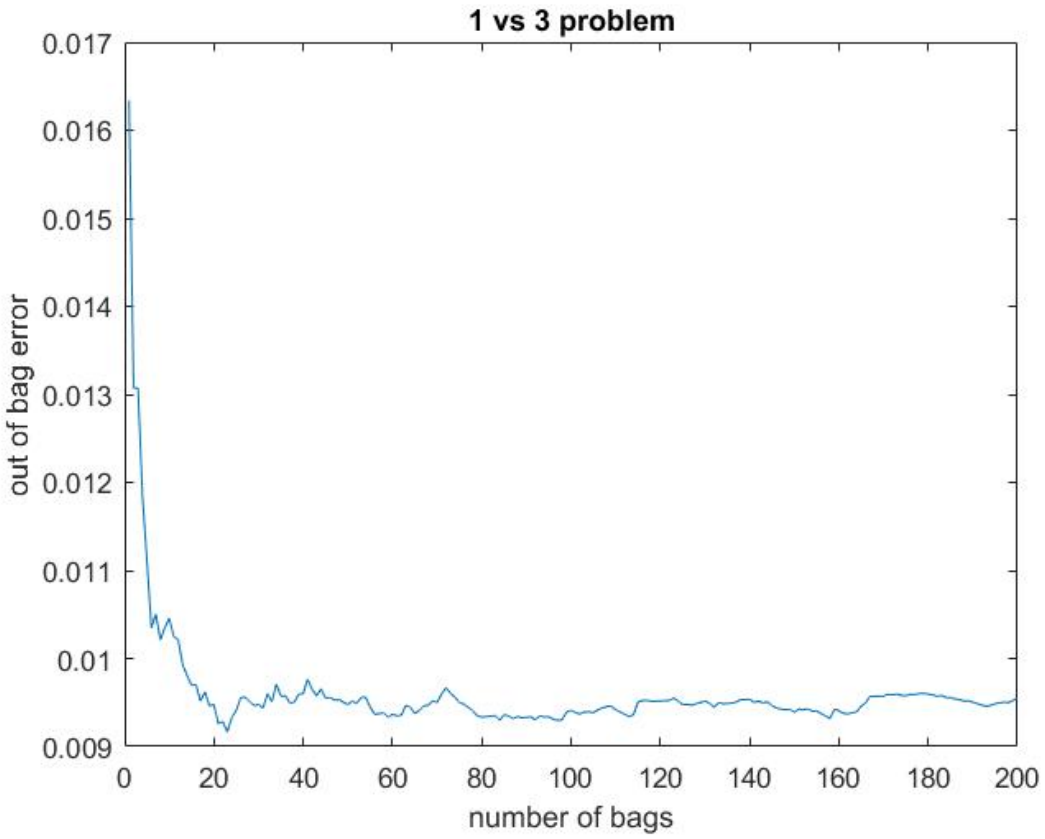
The output for one vs three problem is pasted below from the MATLAB command window

Working on the one-vs-three problem...

The cross-validation error of decision trees is 0.0084

The OOB error of 200 bagged decision trees is 0.0095

The OOB-vs-Bags graph one vs three problem is attached below



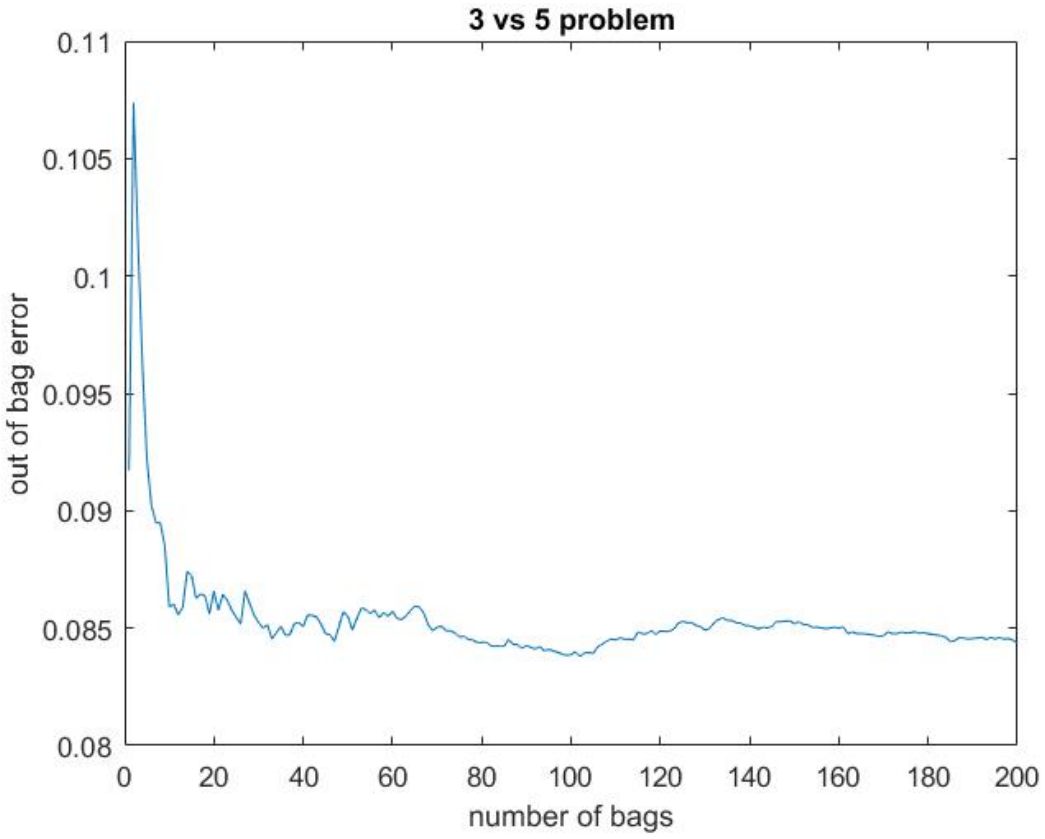
The output for three vs five problem is pasted below from the MATLAB command window

Now working on the three-vs-five problem...

The cross-validation error of decision trees is 0.0626

The OOB error of 200 bagged decision trees is 0.0844

The OOB-vs-Bags graph one vs three problem is attached below



(c) Single decision three and 200 tree ensemble results (zip.test)

Single decision tree performance on testing data

One-vs-three problem: 0.0162

Three-vs-five problem: 0.1197

Ensemble of two hundred trees

One-vs-three problem: 0.0098

Three-vs-five problem: 0.0526

(d) For each tree in the tree ensemble, training data are sampled from the whole training dataset with replacement. Therefore the data points not sampled for training are called out-of-bag observations. The out-of-bag error (OOB) computes the misclassification probability (for classification trees) for out-of-bag observations in the training data[1]. The overall out-of-bag error for the bagged tree is the mean of the OOB for each tree. This is very similar to the cross validation method where the data set is partitioned into training set and test set, the former is used to train the model and the latter is used to estimate the model performance. Likewise, OOB is an estimate of the prediction error of the bagged trees.

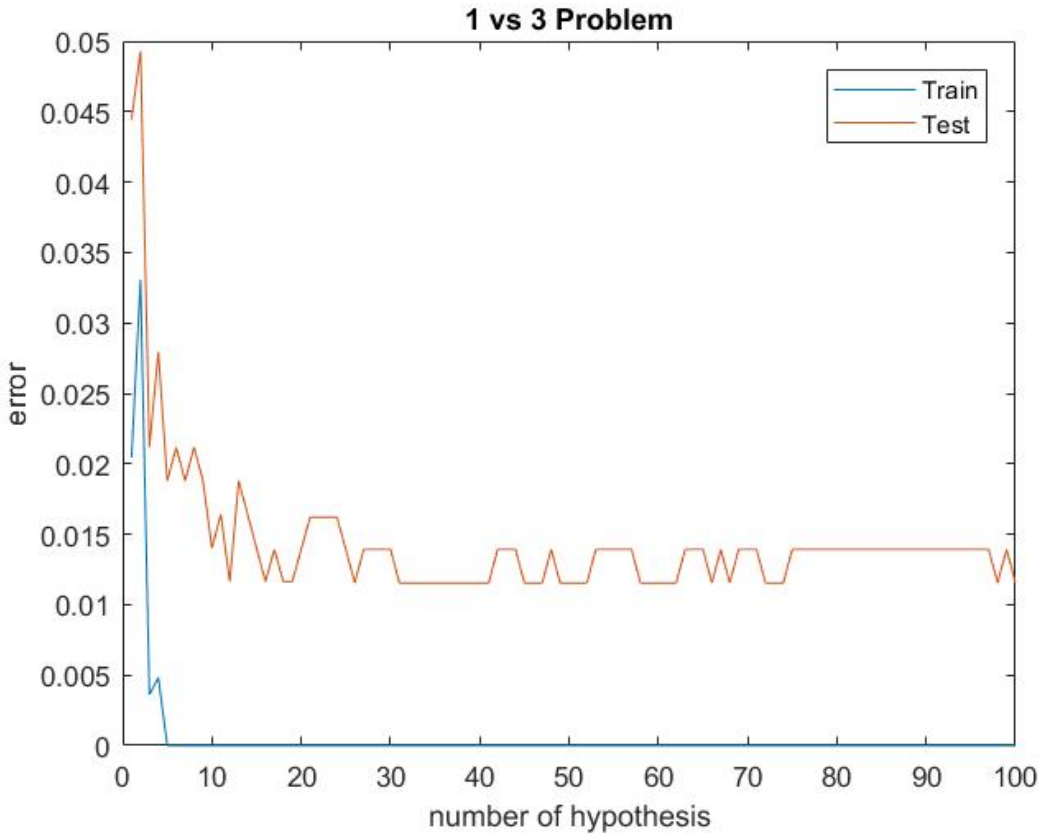
In our training results, an observable trend is that when the number of bags increases, the performance of the overall bagged trees becomes better (smaller OOB). As shown in the graph *1-vs-3 problem*, the OOB decreases from ~0.016 and fluctuates around 0.095. As for the graph *3-vs-5 problem*, the OOB decreases from ~0.11 and fluctuates around 0.085. Those results are closed to the results (cross validation error) from the trees trained with cross validation. This is most likely because the increasing number of trees decreases the chance of overfitting while a single decision tree is very likely to over fit the training data. By add more trees, bagging improves variance by averaging/majority selection of outcome from multiple fully grown trees on variants of training set. Therefore, bagging decrease overfitting and have a better performance than single decision trees. The results from part (c) supports the argument. The ensemble of two hundred trees have an error rate of 0.0098 and 0.0526 for one-vs-three problem and three-vs-five problem respectively, which is much smaller than the error rate of 0.0162 and 0.1197 of a single decision tree.

1. https://www.mathworks.com/help/stats/treebagger.ooberror.html

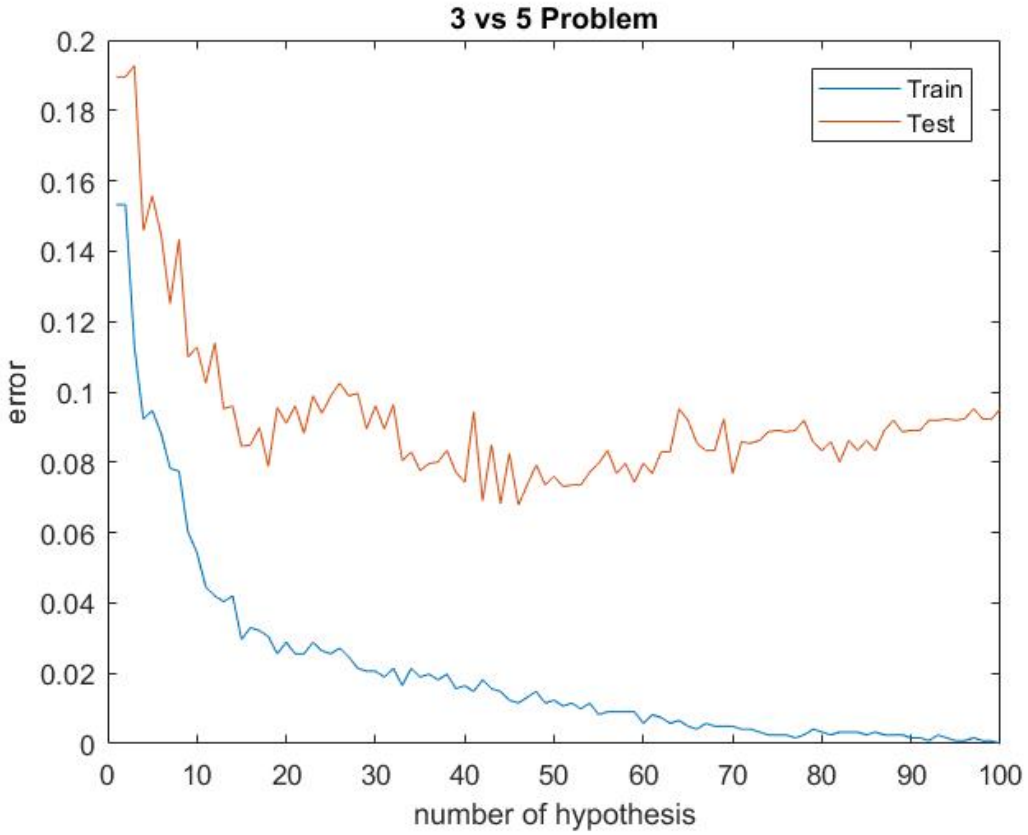
Problem 2

Adaptive boosting, aka AdaBoost, is a learning algorithm that focuses on classification problems and aims to convert a set of weak classifiers into a strong one. The way AdaBoost obtains a strong classifier is that, in each iteration of the training process, AdaBoost adjusts the weights of each training data in favor of misclassified entries. Information obtained from each stage of training the algorithm learns about the relative 'hardness' of each training sample, and the 'hardness' information is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples associated with larger weights. In the end of the learning, AdaBoost will obtain a rather strong classifier.

In the one-vs-three problem, AdaBoost quickly (less than 10 iteration) learns the classifier that achieve 100% correctness for the training set. Shown in the plot below, the error rate in training data decreases as the number of weak hypothesis increases – because previous misclassified data points are now associated with higher weight. The performance of the classifier on the testing data has a similar trend on the plot curve, the error rate levels out to ~0.012, which is very close to the result we obtained in problem 1.



In the three-vs-five problem, starting ~90 iteration the classifier begins to touch the horizontal axis which stands for 0% misclassification, and we can infer from the graph that AdaBoost takes ~100 weak hypothesis to achieve the zero error rate of the training data. Shown in the figures, when the number of weak hypothesis grows, the error rate in training sample decreases – likewise because previous misclassified data points are now associated with higher weights. The performance of the classifier on the testing data is similar – error rate decreases as the number of hypothesis increases – but the error rate begins to level out earlier – starting from twenty weak hypothesis, the error rate fluctuates around 0.85, which is very close to the result we obtained in problem 1.



All in all, the graphs above for the two classification problem supports that as the number of hypothesis increases, the error rate of model trained with AdaBoost will decrease.