Implementation 3 **Group Members**

Derek Jackson
Meredith Leung
Dylan Sanderson

Introduction

This report discusses the configuration and accuracy for multiple decision tree based machine learning algorithms to classify mushrooms as poisonous or safe to eat. The algorithms tested include the classic Decision Tree, Random Forest, and AdaBoost. The Random Forest and AdaBoost algorithms utilize the implementation of the regular Decision Tree as a building block. The regular decision tree (part 1) was trained at varying depths, and results in perfect classification at a depth of 6. The random forest (part 2) reached its highest accuracy when trained on 50 features and with 15 trees. We were surprised that the random forest could not achieve perfect classification, but perhaps we needed to train on more features and/ or more trees. The AdaBoost algorithm (part 3) achieved the highest accuracy with 6 base learners, and a tree depth of 2.

Part 1: Decision Tree

A flexible algorithm for creating a decision tree given a set of input data was created. The decision tree assumed binary splits, and selected features based on the benefit. The benefit was calculated as:

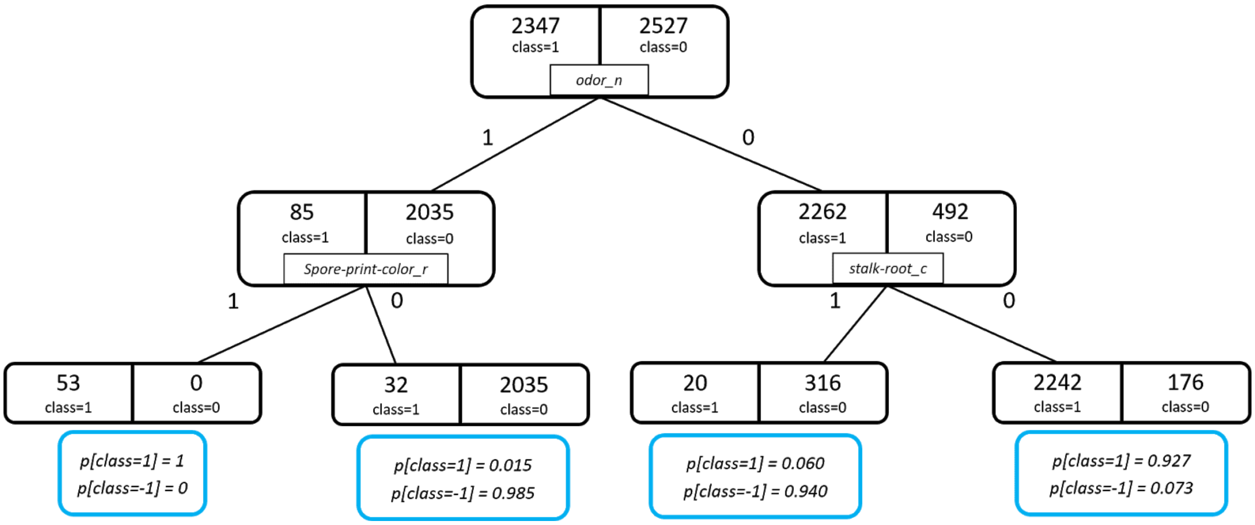
$$B = U(A) - p_1 U(A_1) - p_0 U(A_0)$$

Where p_i represents the probability of being on each node, and U is an uncertainty measure. Here, the uncertainty was computed according to the gini index:

$$U(A_1) = 1 - {p_{11}}^2 - p^2{}_{10}$$

Where p_{11} is the probability that the predicted output is 1 given that the feature split on is 1. Conversely, p_{10} is the probability that the predicted output is 0 given that the feature split on is 1. $U(A_1)$ is computed in a similar fashion. For a given node, the algorithm computes the benefit of the split for each feature. The feature that returns the largest benefit is selected as the feature to perform the split on. This process is repeated until either no more splits are possible, or a pre-specified maximum tree-depth is reached.

(1a) Figure 1 shows the resulting tree when the maximum tree depth is set to 2. The feature that each node is split on is shown at the bottom of the node. The number of features classified as poisonous (class=1) or nonpoisonous (class=0) are shown at each node. It can be seen that following down node 1-1, the tree can perform no more splits; whereas the other nodes can be split further (1-0, 0-1, and 0-0). **(1b)** A tree with a depth of 2 results in a training accuracy of 95.3%, and a validation accuracy of 96.5%. Both indicating a good fit.



(1c) Figure 2 shows the training and validation accuracies when the tree has maximum depths ranging from 1-8. As expected, the accuracy monotonically increases with the depth until the tree perfectly fits the data (depth=6). Interestingly, the validation accuracy is larger than the training accuracy for all depths except those that fit the data perfectly. One possible explanation is the sample sizes of the training (4,874) and validation (1,625) data. If there were more validation data it is likely that the validation accuracy would decrease. This could also just be a coincidence of the validation data selected. **(1d)** A depth of 6 gives the best validation accuracy (100%), and it is not necessary to go beyond this depth.

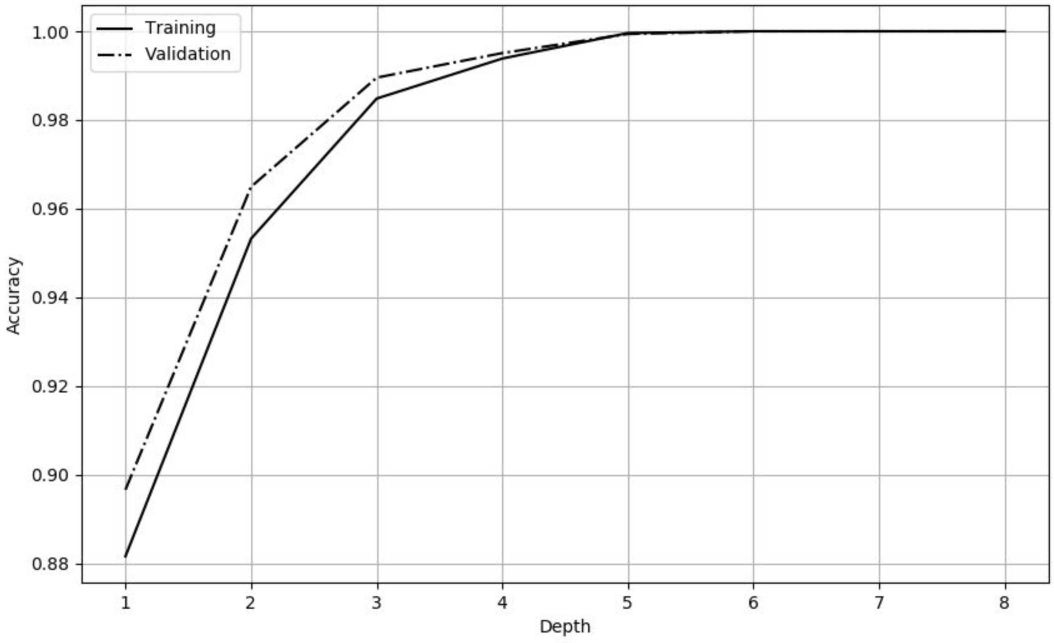


Figure 2: Accuracies of training and validation predictions *vs* the depth of the tree.

Part 2: Random Forest (Bagging)

In the random forest algorithm, we test the dependency of model skill on the number of trees (*n*) in the forest, the number of features for each tree (*m*), and randomness. **(2a)** Each tree had a depth of two and was generated from a new dataset of length 4874 sampled with replacement from the training data. Sub-samples of *m* features each generated their own predictions of the output, and the mode or majority vote of these outputs was taken as the final prediction. **(2b)** Figure 3 shows how the number of trees impacts model accuracy. **(2c)** Adding more trees improved our model's ability to accurately predict whether mushrooms were poisonous or not. There was a dip at two trees, which was surprising, as we expected the model skill to monotonically increase with the number of trees (there is no risk of overfitting as total number of trees in the forest goes up). We interpret this as an indication that the majority vote from two trees may not always predict accurately if the predictions diverge.

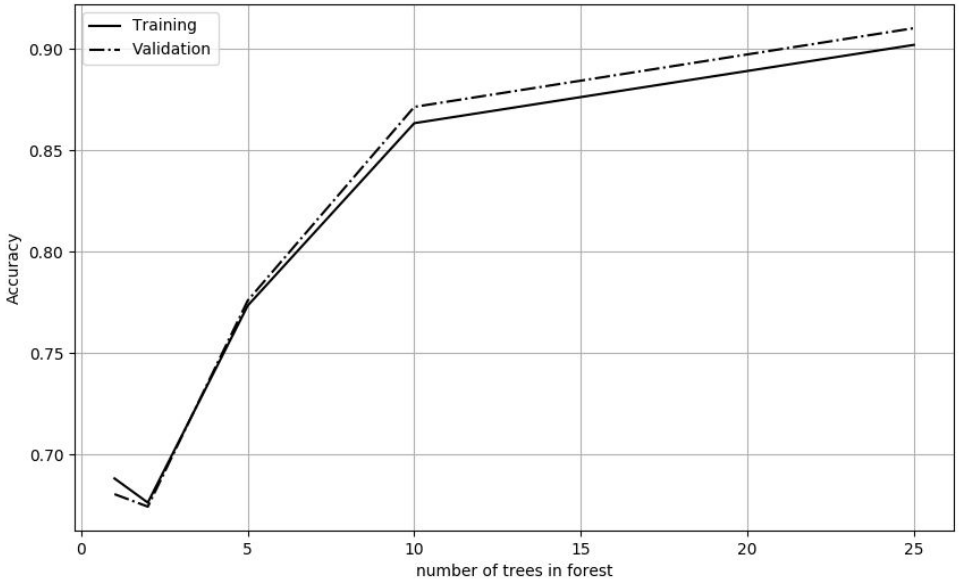


Figure 3. Accuracy of the random forest model as number of trees increased. Forests of size 1, 2, 5, 10, and 25 were tested.

(2d) We then tested how model parameter *m* (number of features on each tree) impacted our train and validation accuracy (figure 4). As the number of features increased, so did our models ability to accurately predict whether the mushrooms were poisonous or not. The figure shows that our model skill increases monotonically, this however may be deceiving, as in general training on too many features leads to overfitting. This indicates we likely could have increased the number of features we trained on beyond 50 and continued to increase our model performance.

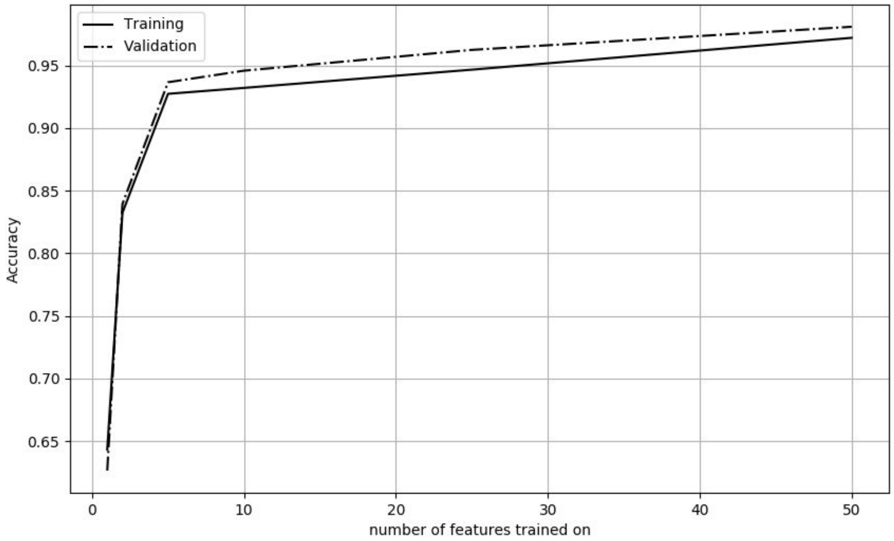


Figure 4. Random Forest sensitivity to number of features trained on.

Using the most skillful model determined through parts 2b and 2d (50 features with 15 total trees), we ran ten trials with different random seeds to test how much stochasticity and shuffling of our data affects model performance. While there is some variation, all of the seeds perform relatively well (> 93% accuracy). So while choosing a random seed can be important, in this case our model is not as sensitive to randomness as it is the number of features or trees.

Random seed	Training accuracy	Validation accuracy
1337	0.975	0.985
1338	0.971	0.980
1339	0.973	0.982
88	0.977	0.985
8888	0.967	0.978
4444	0.959	0.967
13	0.976	0.983
7	0.967	0.980
2	0.954	0.968
9999	0.938	0.954

Table 1. Training and validation accuracies for 10 random seeds. Highlighted line indicates model used to predict on test data.

Part 3: Adaboost (Boosting)

An AdaBoost algorithm was built which utilizes multiple weak learners that will make a decision by a weighted vote from each learner (classifier). **(3a)** Using a weight parameter D that is updated by every learner, each proceeding learner will focus more on the incorrectly predicted training examples of the previous classifiers. **(3b)** A parameter L was included to specify the number of base learners to use in the algorithm. **(3c)** Table 1 includes the training and validation accuracies for a set of base classifiers and specific depth.

# of base classifiers (L), Tree depth	Training Accuracy	Validation Accuracy
L = 1, depth = 1	88.2%	89.7%
L = 2, depth = 1	88.2%	89.7%
L = 5, depth = 1	94.5%	95.8%
L = 10, depth = 1	98.1%	98.6%
L = 15, depth = 1	97.6%	98.2%
L = 6, depth = 2	98.4%	99.0%

Table 2: Accuracies of training and validation predictions for various number of classifiers and depths.

(3d) A significant increase in accuracy was found when increasing the parameter L (number of base classifiers), to a certain point. Increasing L changes the number of weak learners that vote to determine a mushrooms classification. Additionally, each learner will focus on correctly classifying training examples that were previously misclassified. So with more weak learners in the AdaBoost algorithm, there is more opportunity for misclassification correction. This understanding is supported by the accuracies found in Table 1.

L=1 and L=2 ended up having the same training and validation accuracies. This is likely due to how parameter D is updated for each example. After each weak learner is built, parameter D updates only marginally. Thus multiple learners are required to make a significant difference. While it is not clear why the algorithm did better with L=10 and L=15 for depths of 1, one possible explanation could be adding additional weak learners after a certain L will cause the algorithm to 'oscillate' around the optimal decision tree of depth 1.

(3e) Parameter L set to 6 with a depth of 2 provided the greatest accuracy for both the training and validation data, scoring better than both L=10 and L=15 with depths of 1 (see Table 1). This improvement is likely due to the increased accuracy of a decision tree of depth 2 versus a depth of 1. While AdaBoost is good at combining multiple weak learners to improve the overall classification accuracy, it is still limited to the effectiveness of the weak learner. Adding one more layer of decision making is shown to provide greater accuracy than just increasing the number of voting weak learners.

(3f) A prediction file for **pa3_test.csv** was created using the AdaBoost algorithm with L=6 and depth=2.

Discussion of Results and Summary

The results indicate that overfitting is a concern with regular decision trees. For the data set given, a perfect classification occurred at a maximum tree depth of 6. Interestingly, the validation data resulted in higher accuracies than the training data. This could be caused by either the sample size or selection of the validation data. In order to avoid overfitting and build a more robust decision tree, either random forest or AdaBoost can be implemented. The random forest did not achieve perfect classification. While you have to be cautious about overfitting random forests by using too many features, since we did not see any indication of overfitting with 50 features we likely could have raised both the number of trees and features used to train our model to achieve more accurate predictions. AdaBoost algorithm may provide better classification with more base classifiers and trees of depth greater than 1 (as suggested by the results of L=6, depth=2). However, this is beyond the scope of this assignment.