CS534 Implementation Assignment 3

November 25, 2019

Introduction

In this assignment, we used decision trees and ensemble methods to classify mushrooms as poisonous or edible. The mushroom dataset has 4874 training samples and 117 binary features, which were created from 22 categorical features. We disregard one of the features, veil-type_p, because it has only one value for all samples and, therefore, cannot be used for classification.

This report is organized as follows. Part one implements a decision tree and evaluates the effect of tree depth on accuracy. Part two compares the accuracy of random forests with different numbers of trees and different numbers of features considered when training an individual tree, as well as the effects of randomness. Part three compares the accuracy of Adaboost with different numbers of weak classifiers.

Part 1: Decision Tree with Varying Depths

In this section, we implemented a standard decision tree. At each time step, we pick the feature with results in the lowest gini-index as our discriminating feature.

First, we run our decision tree algorithm on the training and validation data, varying the maximum allowed depth of the tree from 1 to 8. The results are shown in Figure 1.

For $d = 2$, we obtain training and validation accuracies of 95.3% and 96.5% respectively. Some key features to note about the observed trends:

- We achieve 100% accuracy for both training and validation at depth 6. This point represents the maximum for the validation data. There's no point in increasing the tree depth further past this point because regardless of the maximum depth, the tree will always terminate when it classifies all training examples perfectly; because the algorithm is deterministic, it will always stop after reaching a depth of 6.
- Interestingly, the validation data accuracy is higher than the training data for all depths; the decision tree does not seem to overfit at all. This trend likely reflects that the validation dataset is qualitatively similar to that of the training dataset. The best validation accuracy, like the best training accuracy, is at $d = 6$.

Part 2: Random Forest (Bagging)

In this section, we implement a random forest. Each tree is trained on a data set of 4874 samples (created by sampling with replacement) and a subset of features (created by sampling without replacement). The random forest has the following parameters:

- n: The number of trees in the forest
- m: The number of features randomly selected when training a single tree
- d: The depth of each tree in the forest

First, for parts (b) and (c), we try varying the number of trees in the forest. Figure 2 shows the training and validation accuracy for random forests with $d = 2$, $m = 5$, and $d \in [1, 2, 5, 20, 25]$. Adding more trees largely increases the training and validation accuracy, which is expected because more trees means that more data samples and more features are considered more often, increasing the overall diversity of the forest.

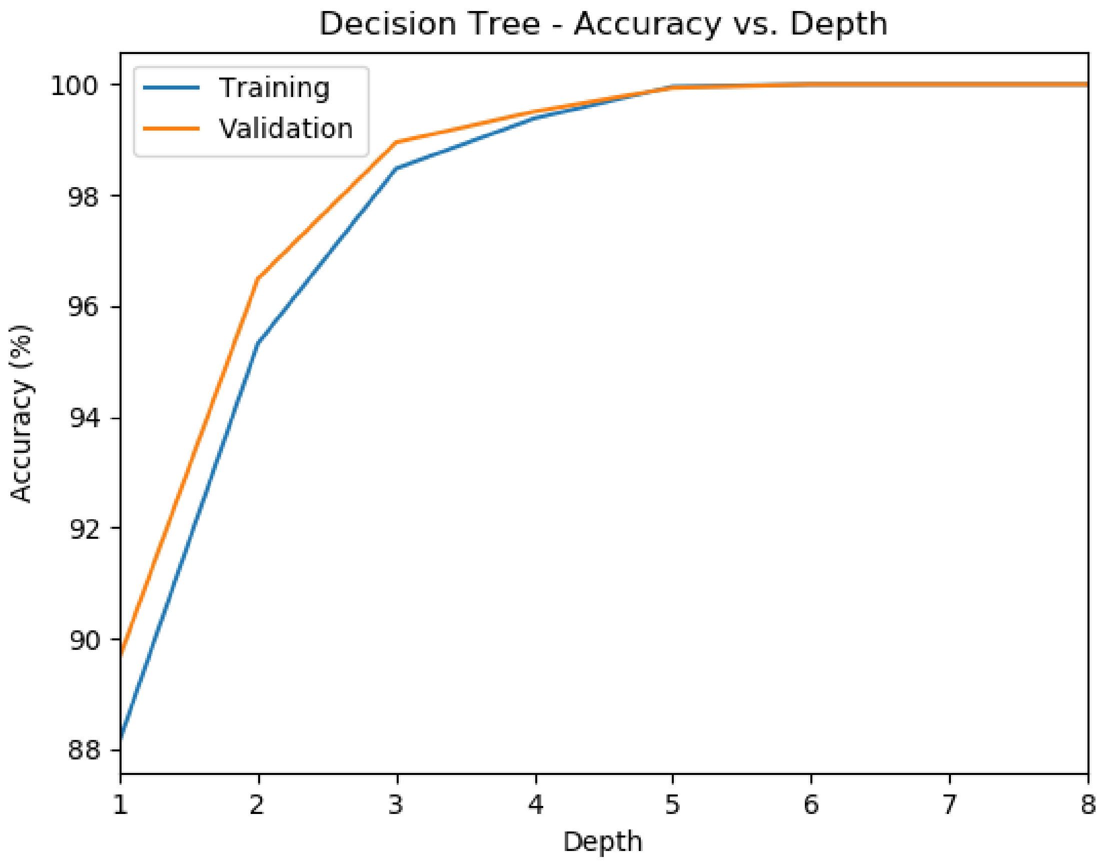

Figure 1: Training and validation accuracies for Part 1.

$n = 2$ and $n = 5$ are exceptions to this trend, likely due to randomness; when there are very few trees, a few particularly bad or good trees will have a significant impact on performance.

Next, for part (d), we try varying the number of features that each tree can consider when making a decision. Figure 3 shows the training and validation accuracy for random forests with $d = 2$, $n = 15$, and $m \in [1, 2, 5, 10, 25, 50]$. The training and validation accuracy largely increase with the number of features considered. This trend is likely due to the fact that considering more features allows more informative features to be selected as decisions nodes. $m = 2$ breaks this trend by having especially low accuracy, and $m = 5$ breaks this trend by having especially high accuracy. These exceptions are likely caused by randomness in choosing which features to consider, as randomness can have a significant impact when very few features are chosen.

From the previous results, we found that $d = 2$, $n = 15$, and $m = 50$ resulted in the highest accuracy of the values considered. For part (e), we trained 10 different forests with $m = 50$, $n = 1$, $d = 2$. The training and validation accuracy of the forests is displayed in Table 1. The average training accuracy was 91.34% and the average validation accuracy was 92.06%. This experiment demonstrates that, with the given parameters, randomness can significantly change the accuracy of a forest, because the quality of the trees within the forest can vary significantly across multiple iterations. This experiment had a range of around ten percentage points. However, random forests (generally) contain more than one tree, and averaging over trees (by voting) allows us to reduce the negative effects of randomness. Overall, randomness, specifically the random incorporation of data and features into our trees, causes the performance of individual trees to deviate significantly; however, when we combine a large number of individual trees, the randomness allows us to build forests of low-correlation trees, thus contributing to lower overall variance.

Figure 2: 2b. Accuracy by number of trees n.

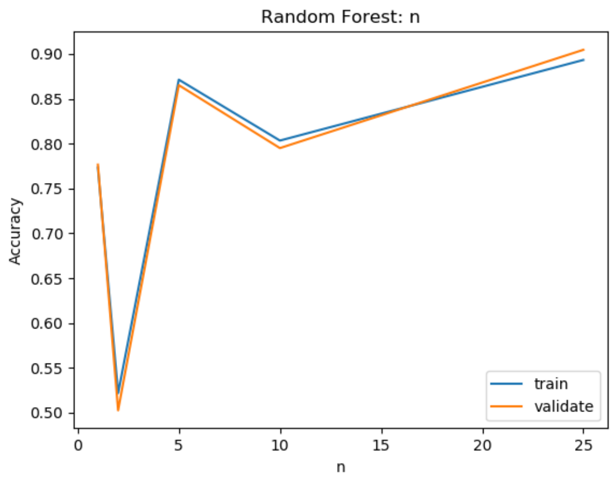

Figure 3: 2d. Accuracy by number of features m.

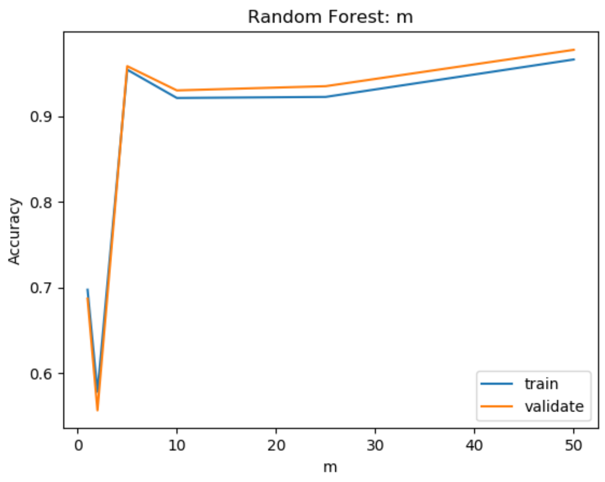

Table 1: 2e. Accuracy of forests with random seeds.

Seed	**Training Accuracy** (%)	**Validation Accuracy (%)**
985	93.89	94.65
883	88.43	90.03
622	91.28	93.05
901	85.37	84.31
500	95.06	95.94
866	93.68	94.52
190	85.37	84.31
638	93.68	94.52
440	95.06	95.94
984	91.63	93.35

Part 3: AdaBoost (Boosting)

In this section, we implement Adaboost by modifying the original decision tree code so that it allows data points to have differing weights; therefore, when considering the gini-index or the decision to be made at a terminal node, instead of p being determined by the proportion of data points belonging to a given data category, we use the sum of the weights within each category.

Figure 4 shows the training and validation accuracy for the Adaboost algorithm where our weak learners are decision trees with a single node ($D = 1$), and L is the number of weak learners we utilize to make our final classification decision (part c). Table 2 has the same information, but in numerical format. In general (part d), we saw that as L increased, accuracy also increased until about $L = 10$, at which point the classification was almost entirely accurate. This is the expected behavior since we would not expect a small number of weak learners to be able to accurately classify a complex dataset; as L increases, the resulting classifier both becomes more discriminative and also focuses on past mistakes made and correcting them. Similar to in Part 1, we did not observe any overfitting.

Finally, we quickly tested what happened if we allowed a slightly stronger classifier to be used in Adaboost, so we tested an Adaboost implementation with $D = 2$ and $L = 6$ (part e). The training and validation accuracies obtained were 98.36% and 98.95% respectively; this is as opposed to the previous $D = 1$ and $L = 15$ classifier, which obtained 99.84% and 99.82% training and validation accuracy. This shows that even though using stronger learners allows us to have more discriminative models at each corresponding L, using a weaker learner is more than sufficient as long as there are a large number of them contributing to the final classification.

L	**Training Accuracy** (%)	**Validation Accuracy** (%)
1	88.16	89.66
2	92.53	94.40
5	97.56	98.22
10	99.84	99.82
15	99.84	99.82

Table 2: Numeric data for Figure 4.

Since our best Adaboost was with $D = 1$ and either $L = 10$ or $L = 15$, we have included a file **pa3_test.csv** which contains the predictions for the test file using $L = 15$.

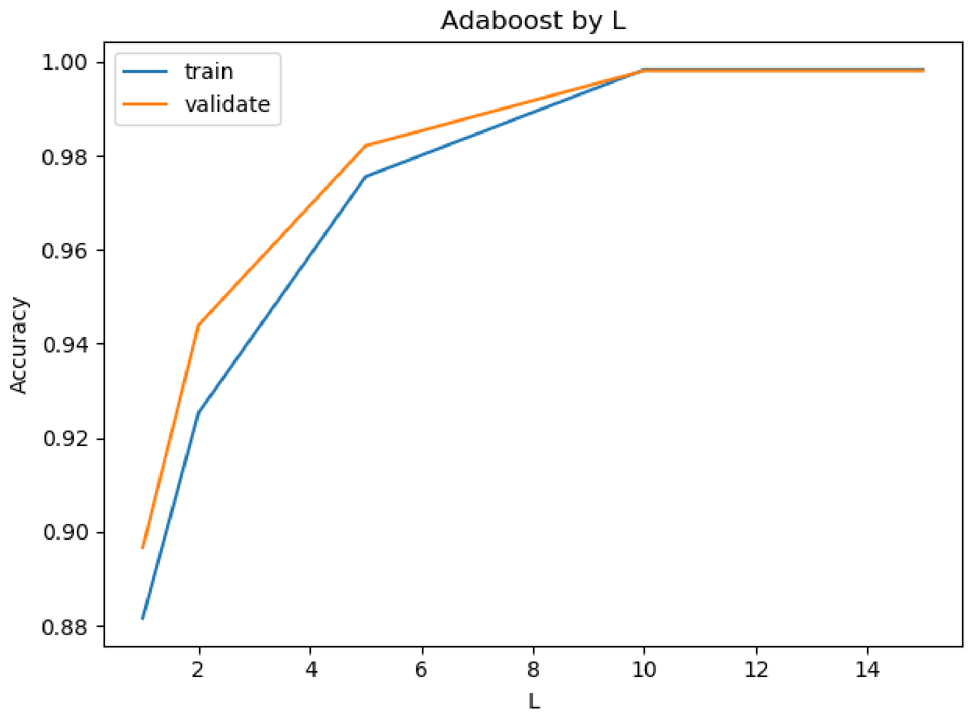

Figure 4: Problem 3c. Accuracy of Adaboost by L