Introduction

In this project, we developed decision tree, and utilize it to create an appropriate model for given data sets. We aims to classify handwritten digits of numbers 3 or 5 using decision tree and compare the performance with the applications of it. We implemented three decision tree algorithms; Decision Tree, Random Forest (Bagging), AdaBoost (Boosting).

In Part 1, we discuss Decision Tree. The Decision Tree in part 1 uses Gini-index to measure the uncertainty. Based on the computed uncertainty, we split the list of all training examples in the way that the uncertainty can be reduced the most. This is called benefit of split for feature f_i. The benefit of split for feature f_i is computed as following;

$$B = U(A) - p_l U(AL) - p_r U(AR)$$

, where A is split in two list AL and AR with f_i at threshold T

Since the given features values in the training examples are continuous, we first search for the best threshold T for each feature f_i, which maximize the benefit. Then, we compare the benefits of each feature, and choose the best feature that gives the highest benefit. We limit the maximum depth of a decision tree to 20 , in which root is at depth 0, in training. We test the decision tree, and record the accuracy at each depth on training sets, and validation sets to compare the performance in different depth. Finally, we report the behavior of the performance against the depth, and show that at which depth the train accuracy reaches to 100%, and we can get the best validation accuracy.

In Part 2, we utilize the decision tree algorithm for Random Forest (Bagging). We implemented Bagging algorithm for given requirements; the number of trees in the forest, the number of features for a tree, maximum depth of the trees in the forest. We set the size of each tree as 63% of the size of given training examples. The data for each tree is drawn at random with replacement among the given 4888 training examples. Such sampled data is sent to decision tree algorithm with *m* number of features drawn randomly from given 100 feature set without replacement. Our random forest algorithm collect such *n* trees as an independent individual classifier, and output prediction using majority vote from the classifiers. We report the train and validation accuracy depending on different number of trees of the random forest, and analyze the tendency. We experiment with different *m* for features drawn randomly from given 100 feature set without replacement, and report how this change affects to the train and validation accuracy.

In addition, we implement AdaBoost algorithm with decision tree to create another ensemble model in Part 3. In AdaBoost algorithm, the decision tree is built with a weight parameter D, a vector of train size. Using this D vector, we focus more on errors from previous classifiers than correctly predicted examples. In each iteration, we measure weighted error, and compute $\boldsymbol{\alpha}$ based on the weighted error, and the number of iteration is decided by L. Such $\boldsymbol{\alpha}$ is used to update and generate the distribution of D_{l+1} based on current D_l. We report the train and validation accuracy for different L, and discuss the behavior of AdaBoost against the parameter L.

Finally, we summarize our results and discuss our concerns and some unexpected results in our experiment in Discussion section.

Part I. Decision Tree

(a) Create a decision tree with maximum depth of 20

We run our program on iMac (late 2015, 3.2GHz Intel Core i5). We treat the first node as the depth=1, so equivalently, the maximum depth at 20 should be 21 in our program. Thus, in total we used 490.5 seconds for running the algorithm. Figure 1. shows the result of building decision tree under the given condition.

```
[0.7371112929623568, 0.8078968903436988, 0.8197626841243862, 0.8692716857610474, 0.9101882160392799,
0.9306464811783961, 0.9519230769230769, 0.9648117839607201, 0.9803600654664485, 0.9883387888707038,
0.9930441898527005, 0.9954991816693944, 0.9973404255319149, 0.9981587561374795, 0.9991816693944353,
0.9997954173486089, 1.0, 1.0, 1.0, 1.0, 1.0]
[0.7286678944137508, 0.7955801104972375, 0.8250460405156538, 0.8520564763658687, 0.8821362799263351,
0.8907305095150398, 0.9023941068139963, 0.9208103130755064, 0.9257213014119091, 0.9226519337016574,
0.9244935543278084, 0.9244935543278084, 0.9226519337016574, 0.9201964395334561, 0.9201964395334561,
0.9201964395334561, 0.9201964395334561, 0.9201964395334561, 0.9201964395334561, 0.9201964395334561,
0.9201964395334561]
0.9257213014119091 9
[Finished in 490.5s]
```

Figure 1.

(b) Plot the train and validation accuracy versus depth

Figure 2 shows the test results on train and validation examples.

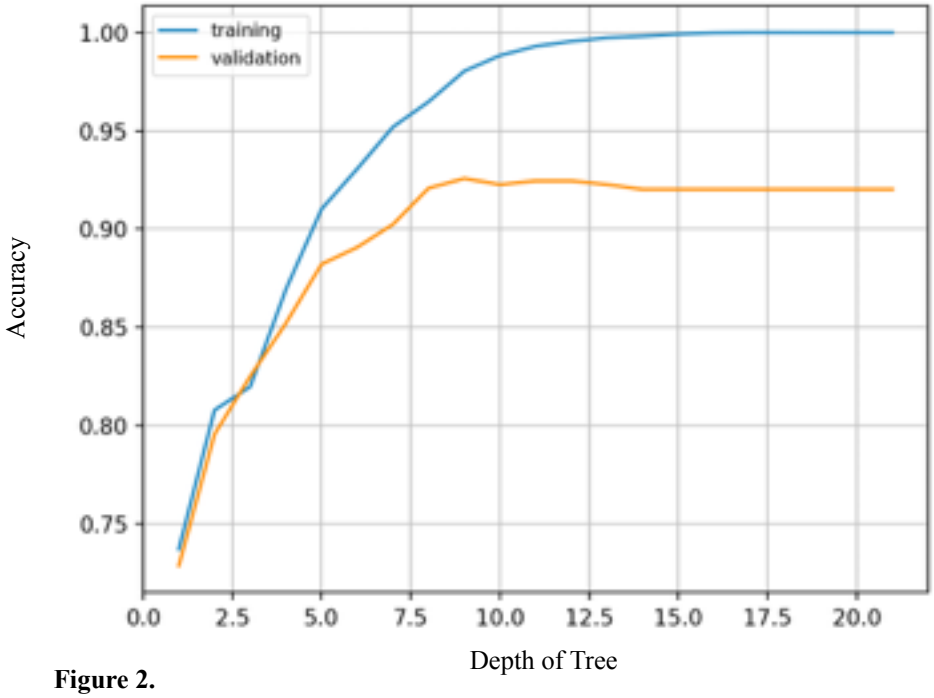

Figure 2.

(c) Behavior of train/validation performance against the depth

The accuracy of training data is increasing monodically. Then it reaches to 100% accuracy when the depth is 17. Because the decision tree (DT) is trained on training set, all of un-classified data in training data could be classified correctly with the increase of the depth.

Surprisingly, when the depth=3, the validation accuracy even higher than training accuracy. We will discuss this situation at the end of the report.

Then, for the accuracy of validation data with the increase of depth, it increase at the beginning of increase of the depth. Then, after at the peaking of depth=9, it falls back a little bit. Then keep stable when the accuracy of training data reaches to 100% which makes a fixed DT.

The overfitting problem exists indeed in this domain due to the fall back after reaching the peak. However, another surprising is the the serious of overfitting problem is not as import as other domain (like linear regression on predicting house price.) The accuracy only falls back 0.5% from the peak. One

of this reason should be DT might be perfect model in this domain which can decrease the effect of overfitting problem as much as possible. We also will discuss this at the end.

Overall Figure 2, we can find that the accuracy at the first a few increase of the depth is increasing really fast. And it only reaches the peak at the 9th depth.
I think the maximum depth may depend on the size of data and the size of feature. Because, there are only 4888 data, so roughly calculating, we could just use 2^15 branches to split it. By adding the first depth of the first node, the roughly estimation depth of 100% accuracy on training data is also 16 which is really close to the real 100% accuracy at depth 17. That is because some feature were using same feature for splitting which may cause repeat nodes separate twice.

(d) Report the depth that gives the best validation accuracy

According to the result, the best validation accuracy is 92.57%, when the depth of DT is 9.

Part 2. Random Forest

(a) Implement a random forest

Figure 3. shows the source code of forest collecting n trees based on sampling data for each tree at random with replacement, and pick m features without replacement using the set variable '`used`'.

```
    for i in range(n):  # n trees
        index = []
        while len(index) != length: # sampling data with replacement
            index.append(choice(range(4888)))
        used = set()
        featureIndex = []
        while len(used) <= (m-1): # sampling features without replacement
            t = choice(range(100))
            if t not in used:
                featureIndex.append(t)
                used.add(t)

        for elem in index:
            data[i].append(getArow(alldata,elem,featureIndex))
            y[i].append(ally[elem])

    forest = [[] for i in range(n)]

    for i in range(n): # collecting n trees to forest
        forest[i] = dt(data[i],y[i])
```

Figure 3.

(b) Plot the train and validation accuracy versus the number of trees in the forest

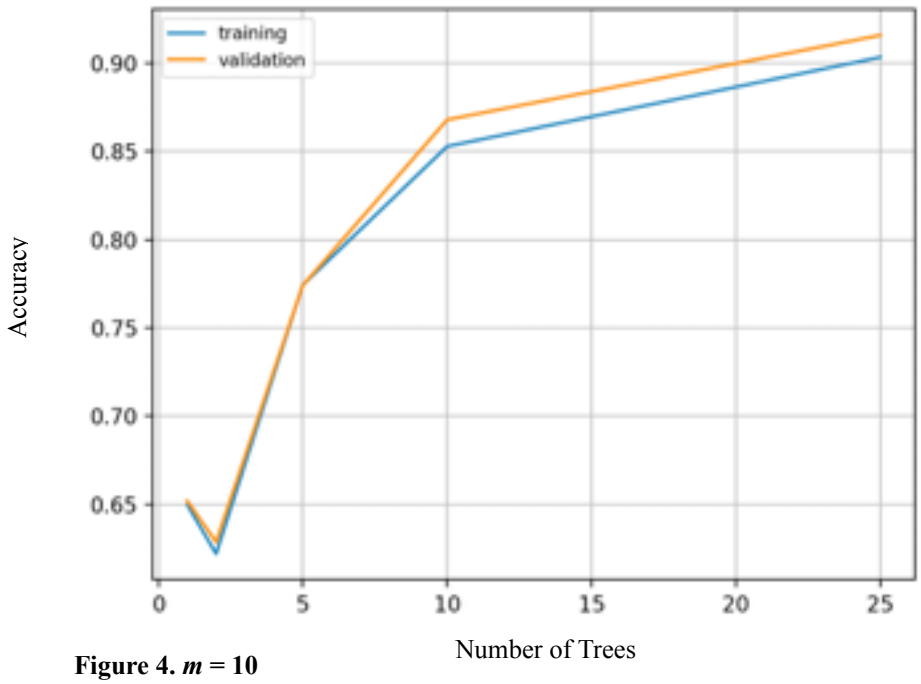

Figure 4. $m = 10$

(c) Effect of adding more tree into a forest on performance

Both train and validation accuracy continuously increase with the increase of n, the number of trees of a forest. Figure 4. shows the result of our experiment with depth = 9, $m = 10$, and $n \in [1,2,5,10,25]$. The validation accuracy shows better performance in the range where $n \leq 10$, which is unexpected. It also shows that when n = 2, the performance of both train and validation accuracy drops to 62% from 65% when n = 1. We argue that this happens due to the random function we use for sampling as well as the number of trees. This is discussed in Discussion section with other results of same experiment.

(d) Repeat above experiments for $m \in [20,50]$

Figure 5. shows the result of the above experiments but with $m = 20$. It reports lower accuracy on both train and validation data overall, but when n = 25, it reaches to over 90% accuracy, which is same as the result of m = 10. Also, it is found that our random forest algorithm with $m = 20$ gives more stable tendency in increasing. In Figure 4., we can see the unexpected drop when n = 2. However, when $m = 20$ and $m = 50$, it does not happen. Figure 6. is the result of the same experiment with $m = 50$. The overall

low accuracy problem is not caused in this case, but the tendency of increasing is not stable as much as *m* = 20. This gives us the view that when *m* is too big or too small, it can also bother to build a good model.

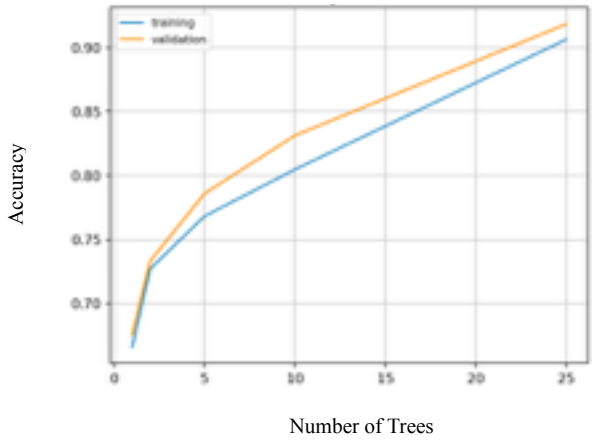

Figure 5. *m* = 20

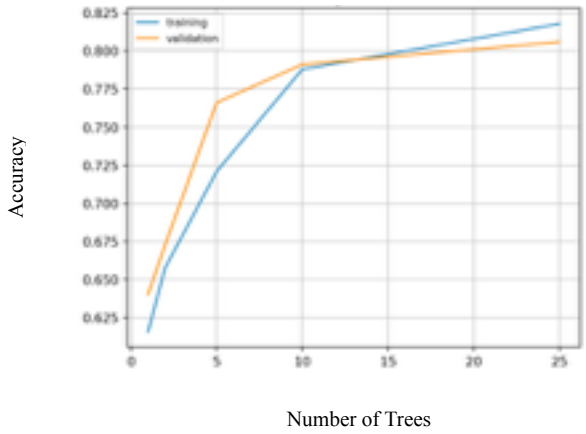

Figure 6. *m* = 50

Part 3. AdaBoost

(a) Implement a decision tree with D

Due to the different sequence of the depth of first node, the depth of weak learner of our algorithm is 10 (cuz we treat the first node as the depth=1). Then, before both training and testing process, we scale the data with parameter vector D which is also has a list which stores all different parameters for different tree. Then we have a forest list which collects all different models at different stages. Thus, after training, we could use original tree to give a prediction and use alpha vector to give a weight and summarize whole result from different trees.

(b) Develop the AdaBoost algorithm with parameter L

In algorithm, we found that with the increase of L, the running time of algorithm increase a lot. For decreasing the running time and trying some favor of AdaBoost algorithm, we tried two different version of AdaBoost.

(c) Behavior of train/validation performance against the depth

For two versions of AdaBoost, the first one showing above is running the algorithm on full size of the data. It took really a long time for the finish of the algorithm. The second one shows blow is only using the first 2000 rows training data to train the model. We can see that the result for that is pretty good. However, due to lack some data, the model gets easier to be overfitting on that 2000 rows training data. So after this model gets overfitted on 2000 data, the increase of accuracy of training data gets slowly. While, we still can see the increase of accuracy of validation. The accuracy for that is pretty good with over 90% with only the first data of that, which is really impressive.

L	1	5	10	25
training accuracy	87.60%	93.80%	95.15%	too long
validation accuracy	87.53%	89.20%	90.30%	too long

Table 1.

Figure 7.

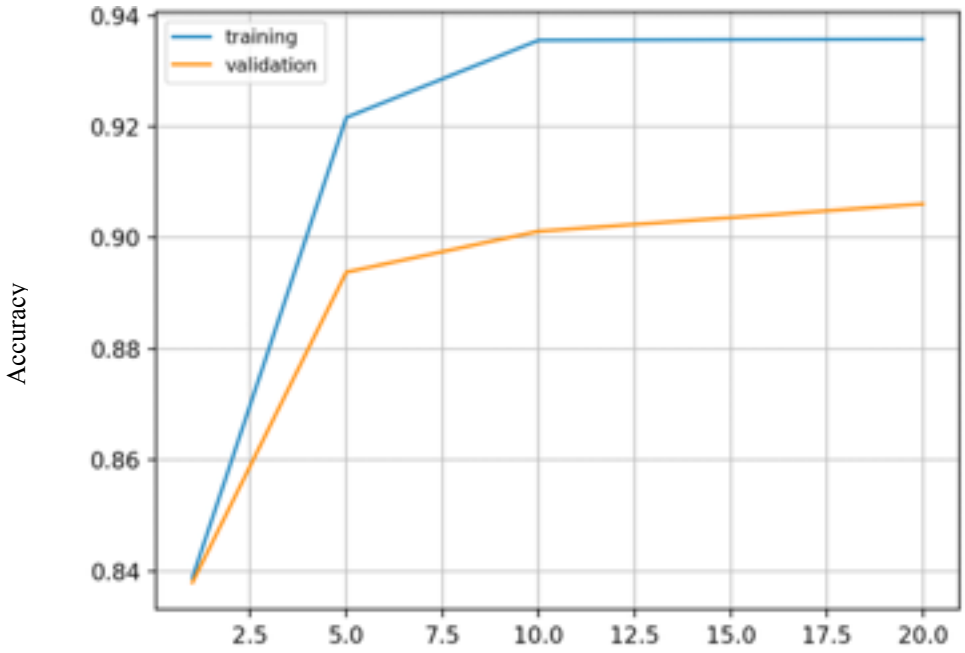

Figure 8.

(d) Behavior of AdaBoost against parameter L

When the L is increasing, the running time is increasing a lot. It took a few minutes on L = 5, however, it took much longer with L = 10. Then, with both results we generated, it is not hard to see that the increase of accuracy rate is almost monotonously increase. For the both of these two pictures, the accuracy rate on both training data and validation data are increasing monotonously. After L from 10 to 20, the accuracy rate of our second trial on training data almost stop growing due to the overfitting issue. But the overfitting issue is not as severe as overfitting issue in other model or algorithm. Like what introduced in slides, the control of overfitting problem is quite well. But from our current experiment result cannot dress this conclusion. Because, in part I, we also observed the slight overfitting issue. Thus, this slight overfitting issue may have three reasons: First, the weak learner of DT provides a good control of overfitting issue. Second, this AdaBoost prevent overfitting issue indeed. Last, so far, the L we tested might be too small for a overfitting issue. The behavior of this algorithm is also quite like the process of human being. People always wants to make fewer and fewer problems and gets experience from what they did or what they got wrong. Thus, the data could be treated as our life experience, after using alpha to update the experience, which can provides us a good result intuitively and practically. To summarize, when the L is getting bigger, there would be overfitting problem indeed, however the influence of overfitting problem seems not severe. Then when the L is getting bigger, both training accuracy and validation accuracy could increase well, which means the performance of AdaBoost did really well on

predicting. At last, when the L is getting bigger, the running time seems increase exponentially or polynomially.

Discussion

Our decision tree algorithms shows that the train accuracy increases monodically with increase of depth, and reaches to 100% accuracy on training examples at depth 17. On the other hand, the validation accuracy increases with the increase of depth at the beginning of increase of the depth, but since the depth gets to 9, it falls back a little bit, and becomes stable at the point where the train accuracy reaches to 100%. It costs 490.5 seconds running time for creating decision tree with depth 20.
Surprisingly, in decision tree algorithms, when the depth=3, the validation accuracy even higher than training accuracy as we mentioned in **Part 1. (c).** The accuracy of validation is slightly higher than the accuracy of training data. Intuitively, it should be lower. However, our explanation for that is for two reasons. The first explanation is the similarity between the validation data and training data may is too high. As a result, when the model of DT gets overfitting and reaches at 100% accuracy rate, the accuracy of validation data set just decrease a little bit and still reach to over 92% accuracy rate. The other explanation is the sparse model with depth=3 situation. In this case, the DT is not as modeled by training data as much. With the similarity from the first explanation, it is really possible that the model get more fit one validation data, since validation data is too similar to training data. There exists this probability indeed.

Random Forest algorithms reports that both train and validation accuracy continuously increase with the number of trees in forest. The reported result in Part 2. shows the better performance on validation data in accuracy. We claim that this is caused by random sampling, and the probability to have same votes from trees when we only have two trees. Figure 7. is the other result of same experiment as Part 2. under the same conditions. Figure 7. gives the same increasing tendency of both train and validation data with the number of trees, overall. However, it shows that when n = 2, the performance of both train and validation accuracy drop, same as Figure 4. We consider this as a proof of the issue of the random function we use for sampling. As shown in Figure 7.,

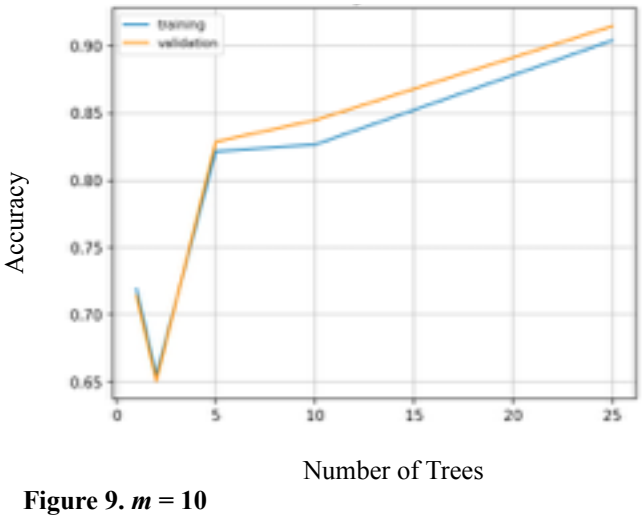

Figure 9. *m* = 10

when we sampling the data for each tree, and *m* features, we pick those at random under the given replacement condition. We believe that this random function affects to the performance by choosing less relevant features since this does not happen with $m = 20$ and $m = 50$, or not enough data for training. Such lack causes unstable results shown as Figure 4. and Figure 7. Also, when $n = 2$, we have high possibility to get +1 from one tree, and -1 from the other tree in majority vote. Such results does not give any clear evidence to choose neither +1 nor -1. We believe that we can avoid this issue using bigger *n*, bigger *m* as Part 2. (d), or weighted vector so that we can choose the one we need to focus more like AdaBoost.

In Part 3, we show that the both train and validation accuracy increase almost monotonously. One issue we had with Adaboost is running time. Running time increases terribly with increase of L. It seems that the running time increases either exponentially or polynomially.