Report for Assignment 3(A) of COL 774

Ayushi Agarwal
2018ANZ8503
ayushi.agarwal@cse.iitd.ac.in

1 Introduction

The goal of this assignment is to implement Decision Trees for classification problems. Random Forest which are a flavour of the decision trees are also explored in this assignment. The assignment is broken down into two parts which are as follows:

- Decision Trees and Post-Pruning - Section 2.1 discusses the implementation of the decision trees using python. Section 2.2 discusses how post-pruning as a method to deal with over-fitting in Decision Trees.

- Random Forest - Section 2.3 discusses how random forests (which are an extension of decision trees) perform on out dataset and then compare them to the decision tree implementation. Section 2.4 discusses the sensitivity of optimum random forest model found from the Grid Search on the parameters including number of estimators, number of features used to split at the node and the minimum number of samples needed to split.

The next section of this report discusses the algorithm in each question, the method to implement the algorithm, report the results and present an analysis of the results obtained.

2 Method, Results and Analysis

In this section, we would discuss our algorithm implementation and the results obtained. We would present an analysis of the results as well[I]. We are going to work on the VirusShare dataset that can be downloaded from online resources. We will be detecting the infected files from this dataset using Decision Trees. Since the dataset is highly sparse, we use pyxclib library to import the dataset.

[I] https://github.com/agarwal-ayushi/Machine-Learning-Assignments/tree/master/Assignment3

2.1 Part A - Decision Tree Implementation

A decision tree is a machine learning algorithm in which the hypothesis is organised in the form of a tree formed by learning certain rules at each node. We learn rectilinear boundaries since only one rule based on one attribute of the data is learnt at a time.

At each node of the tree, we find the best attribute to split the tree into left and right sub-trees. The best attribute is based on the highest mutual information that it holds with the other attributes i.e. maximum correlation with the other attributes. We maximize correlation to divide the data purely. At every node, we split the data based on a threshold and find the mutual information with the other attributes. The attribute split which leads to maximum mutual information is chosen for the current split.

Since the dataset has continuous values of the attributes, we use the median of the attributes to split two-way at a particular node. And hence, the attribute is not removed from the samples, because at another node, another median value of the same attribute could be the one with maximum mutual information.

$$\begin{aligned} \max \mu_I(X,Y) &= \max \mu_I(Y,X) \\ \max(H(Y) - H(Y|X)) & \\ H(Y) &= \sum_{k=1}^{r} p_k \log \frac{1}{p_k} \\ H(Y|X) &= \sum_x H(Y|X=x)P(X=x) \\ H(Y|X=x) &= \sum_y H(Y=y|X=x) \end{aligned} \tag{1}$$

Algorithm 1 shows our methodology to build the decision tree on the training data. The algorithm is to recursively keep adding nodes to the tree by splitting at the median of the best attribute according to highest mutual information which is found as described in Equation 1. $H(Y)$ is the entropy of the labels at a particular node. $H(Y|X)$ denotes the reduction in the error obtained by the current split under test.

We have used a hyperparameter called depth of the tree to build the tree of successive heights and report accuracy of these trees as they grow with increasing depth. With each depth the total number of nodes in the tree will differ and we will be able to analyze the accuracy obtained on the test and validation set on different depth trees. In the code base, we have released the pickle files of all the trees with different depth. The code otherwise will only create a full grown tree.

Algorithm 1: Building Decision Trees

Result: Full-Grown Tree
Repeat (till tree is grown to depth=d) {
1. **IF** the current depth is equal to depth_threshold
   a. Make the current node - LEAF
   b. Assign majority prediction to the LEAF
ELSE
2. **IF** data split at this node is pure (i.e. all samples of one class)
   a. Make the current node - LEAF
   b. Assign the final prediction depending on the class left
ELSE
3. FIND THE BEST ATTRIBUTE from all the attributes in the data to split
   a. Split the data on the threshold(Median)
   b. **IF** all the samples in this split go to the left or right split
      i. Make this node - LEAF
      ii. Assign Majority Prediction
   ELSE
   c. Add New Node based on the best attribute found.
   d. Recursively grow the left tree at this node.
   e. Recursively grow the right tree at this node.
}

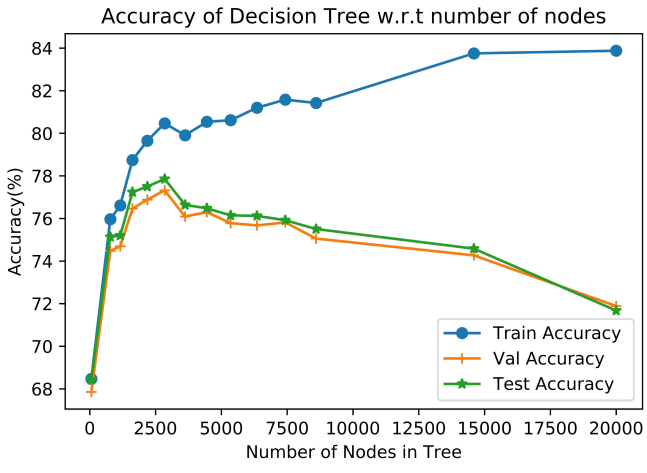


Figure 1: Accuracy Obtained with Increasing Number of Nodes in the Decision Tree

Algorithm 2: FIND THE BEST ATTRIBUTE

Result: Best Attribute to split
Input: Current Node Training samples and train classes
Output: Max. Mutual Info, Best Attribute, Threshold
1. Find Median of all the attributes in the current samples.
2. Choose one attribute at a time
   a. Split the samples at the median of this attribute.
   b. Find Mutual Information as in Equation 1
   c. Check if it is the largest among all the attributes.
3. Repeat 2.

The time taken to grow a full tree is around 13 mins. Figure 1 shows the performance of the decision tree on training, validation and test dataset as we grow the tree of different depths, hence increasing number of nodes. The figure shows that as the tree grows, it starts over-fitting on the training data as the training accuracy increases but the validation and test accuracy starts to decrease rapidly. This happens because we keep splitting the data till we get pure samples at the leaf and hence, the hypothesis becomes less and less generalized and more and more accurate on the training data. We also notice that the over-fitting increases as we grow the tree and the gap between the training and the validation accuracy of the full-grown tree is around 15%.

There are many ways to solve the problem of over-fitting in Decision Trees like:

- Add a hyperparameter: Height or depth of tree to the training model. Build the tree to the height that gives maximum generalization. (We have used to approach in our implementation).
  For growing the tree completely, we set the depth threshold to $+\infty$. Hence, the tree grows till the leaf nodes have pure data.
  In our case, we found the **depth=14** is most optimal achieving a validation accuracy of 77.32% and test accuracy of 77.85% with 2847 number of nodes. After that over-fitting starts.

- Early stopping: Stop growing the tree as soon as the validation accuracy starts dropping. But this can lead to inefficient hypothesis as there might be future nodes existing which could have increased the generalization accuracy.

- Post-Pruning: This is the technique that we will explore in the next section of this report. In this, the tree is grown fully and then pruned till the validation performance keeps increasing.

2.2 Part B - Post-Pruning on Decision Trees

One of the ways to reduce over-fitting in Decision Trees is to grow the tree completely and then post-prune it based on the validation dataset. In this algorithm, we greedily prune the nodes and the sub-trees below it so that the resultant tree gives the maximum accuracy. This is repeated until it starts giving a decrease in the validation accuracy.

A tree can be traversed in many orders and we have iteratively pruned the nodes in the decision trees by traversing in two different orders - Preorder Traversal and Inorder Traversal.

In Preorder traversal, we traverse the root first, then the left subtree and then the right subtree. So the pruning algorithm prunes a complete subtree located at a particular node. In contrast to this, Inorder traversal prunes the left subtree first and then the right subtree.

Result: We observe that the Preorder pruning of the complete sub-trees lead to better accuracy and also overcomes over-fitting. Figure 2b shows the accuracy growth when the tree is pruned by removing complete subtree at a node. The **Validation Accuracy** achieved by the best tree is 79.04% and the **Test Accuracy** is 79.06% with 6821 nodes. Figure 2c shows the accuracy trend as the tree is pruned iteratively by removing its left subtree and then right subtree. The **Validation Accuracy** achieved by the best tree is 78.5% and the **Test Accuracy** is 79.03%.

Hence, we see that removing the complete subtree at a particular node performs better.

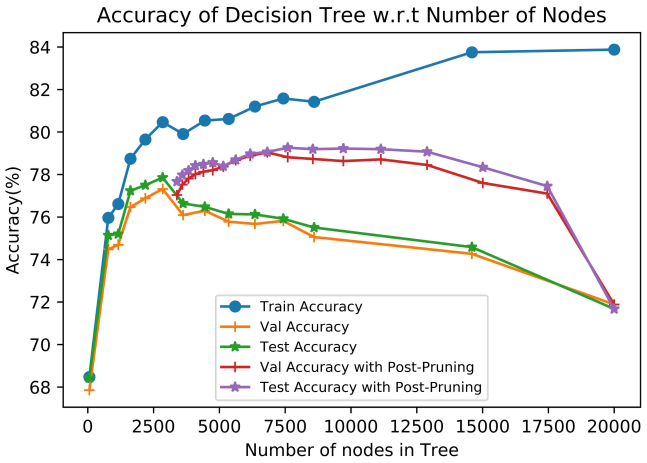
Figure 2a shows the effect of reduced-error post pruning done on the decision tree. The gap between the training and the validation and Test Accuracy has reduced.

2.3 Part C: Random Forests

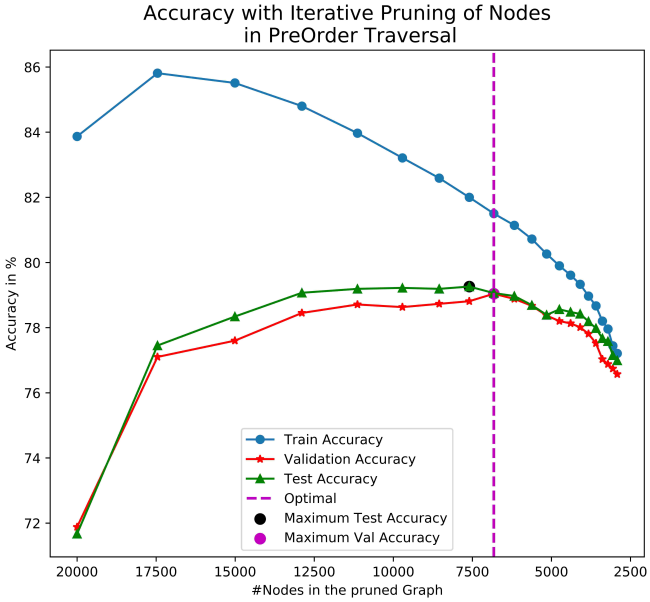
Random Forests are extensions of decision trees where multiple such trees are grown on bootstrapped samples sampled randomly with replacement from the original training set. Several such trees are grown in parallel and then the predictions done on all these trees. We will explore the BAGGING technique in this section and talk about Gradient boosted Trees in Section 2.5.

Bagging: Since we work with a finite set of samples, it could lead to large variance in the model since the Decision Trees learnt on different finite set of samples coming from the same distribution. So, Bagging is used, in which several bootstrapped samples are generated from the same dataset by randomly sampling with replacement. These datasets form a bag of samples on which different Decision Tress are learnt. While doing predictions on new data, the predictions are done on all the trees and the average prediction(regression) or majority prediction(classification) is selected.

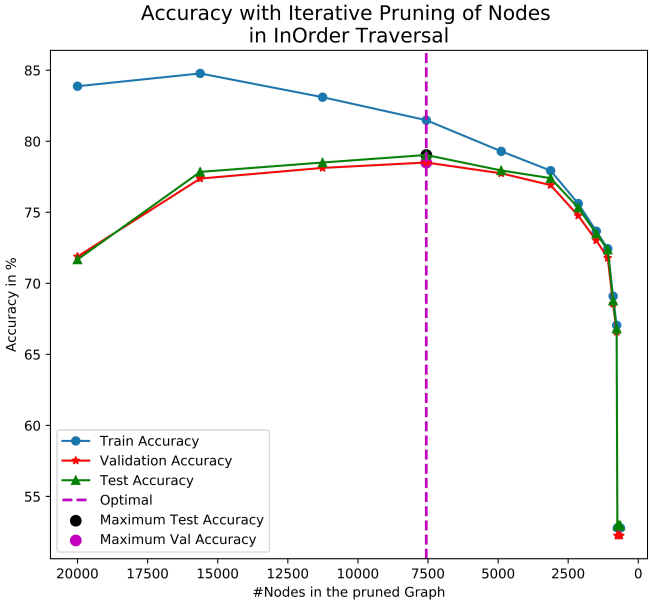
Drop Out in Random Forest: While training all the trees in the random forest in parallel, we don't use all



(a) Post-Pruning when compared to the Decision Tree Growth Accuracy Curve



(b) Accuracy with Iterative Pruning of complete subtree through Preorder Traversal



(c) Accuracy with Iterative Pruning through Inorder Traversal

Figure 2: Iterative Pruning of Decision Trees

the features to train all the trees. Instead, we randomly select certain set of features for every tree and train that tree only on those selected features. This removes the dependence of the trees on a particular feature and also the models generalize better since they learn on limited data.

Out-Of-Bag Error: It is a prediction estimate done on bootstrapped samples. It is the mean prediction error of all the training samples done only on the trees that didn't have those training samples in their bootstrapped samples. Hence, OOB error is equivalent to cross-validation since we don't need to separate the training and validation data to optimize the model.

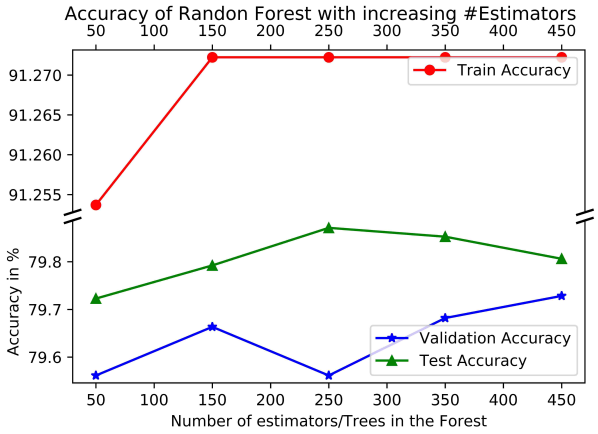
In this Section, we use the Scikit-Learn Python Library called **RandomForestClassifier** to train Random Forests on the given dataset. This library trains a number of trees in parallel according to a number of hyperparameters settings that control the number of trees in the forest, the number of attributes used to learn the tree, using bootstrapped samples or not, etc. We will be using the criteria of 'entropy' to train the Random Forest so that we can compare the results with our own implementation in the previous section. In this section, we will explore three parameters of this library and how the accuracy changes with these parameters.

- **n_estimators:** This parameters shows the number of trees learnt in a forest in parallel. Each tree is fully-grown and there is no pruning. However, there is a parameter called *max_depth* that can be used to experiment with various depth.

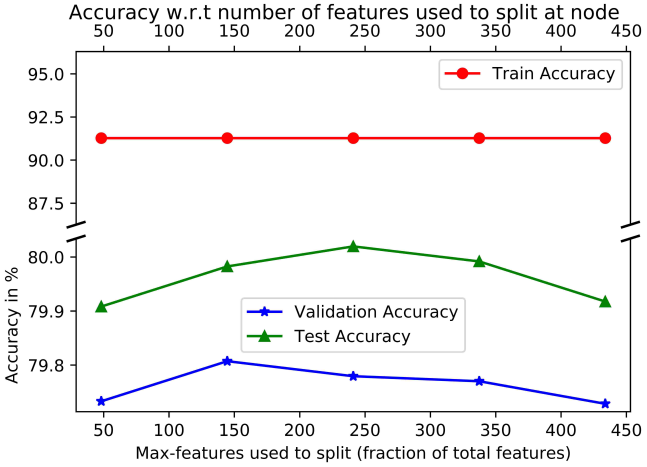
  We tried with depth = 14, and the results were quite similar to our own Decision Trees implementation with depth= 14. The accuracy achieved was within $1 - 2\%$ by our model.
  However, the trees in the random forest should be grown to full depth because they can never overfit due to several other features like using bootstrapped samples and using lesser number of attributes to split.
  Figure 3a shows the Accuracy of the Random Forest Classifier with increasing number of trees in the forest. Only this parameter is varied and all the other parameters are set to default (max_fea is 21). This figure shows that even with the increasing number of trees in the Forest there is no overfitting. But as we will see later in this section, these trends change when other parameters change.

- **max_features:** This hyperparameter is the only parameter to which the model is very sensitive. It controls the number of attributes that are selected randomly to build each tree in the forest. And since the attributes are randomly selected the model behaves differently with different executions with the same values of the parameters.

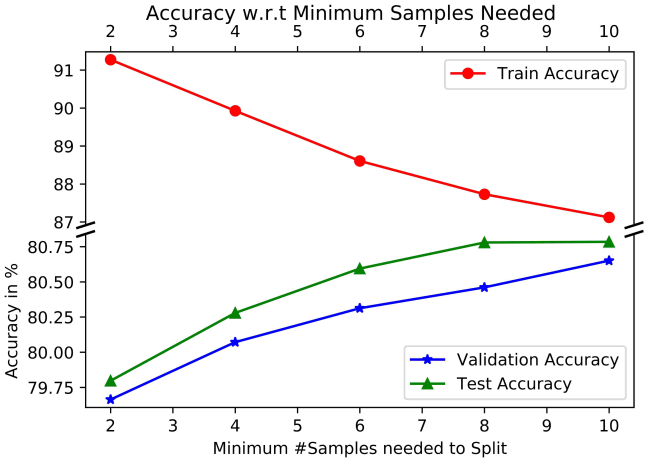
  Figure 3b shows that the accuracy is highly dependent on the maximum number of features used to split the model. The number of estimators used for this figure is 450 and the other parameters are at default. We see that we can achieve higher accuracy than that was obtained by using 21 max features in the previous part and 450 number of estimators.



(a) Accuracy with different number of estimators in the Random Forest



(b) Accuracy with different number of maximum features selected to build tree



(c) Accuracy with different number minimum samples needed to split at a node

Figure 3: Accuracy trends with changing parameters in the Random Forest

- **min_samples_split:** This parameter controls the number of samples that are needed at a node for splitting at that node. As this parameter increases, more and more nodes start becoming leaf nodes before the data is pure. It makes the predictions smoother. Figure 3c shows this trend itself. As the minimum samples needed to split increases, the accuracy of the forest increases since it increases generalization of the forest.

2.3.1 Grid Search over these parameters for finding Optimal Model

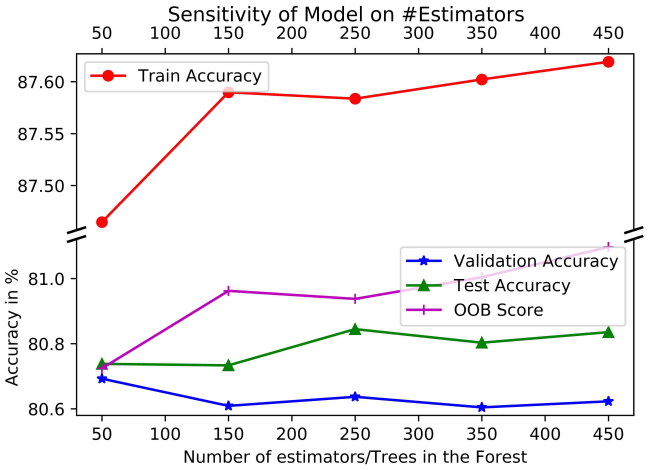
As discussed in Section 2.3, Out of bag error can be used to select the optimal value of these parameters. Random Forest don't need a separate cross validation set to test the parameters since we can use the OOB error. We use the GridSearchCV library from SKLEARN to do a grid search on this parameters using a custom scoring function that returns the OOB score of the model. It also uses kFold cross validation. Even though we don't need cross validation score since we get the model OOB score during the training only, but we still do 5-fold cross validation. The trees are built using bootstrapped samples and it might be possible that the tree doesn't see almost one-third of the data. Creating kFold cross validation samples, we make sure that the model sees all the samples atleast once. So we have used 5-fold cross-validation. Higher the number of folds, greater are the chances that the model will see almost all samples atleast once.

Results: We found different optimal parameters by using the scoring function of 'Accuracy' vs. 'Out-of-Bag Error'.

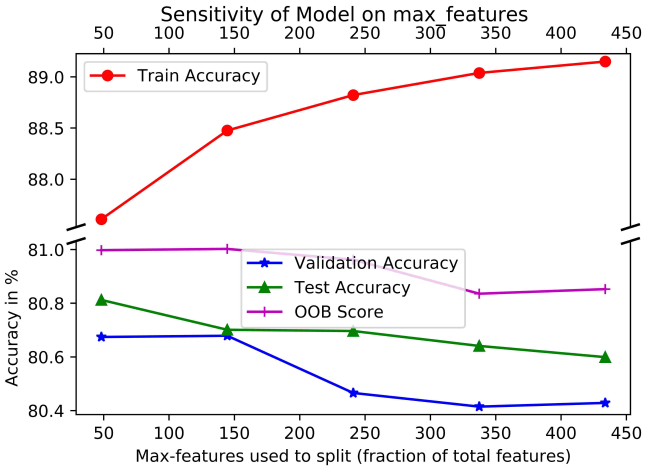
- **Scoring Function - Accuracy**
  Optimal parameters - {'max_features': 0.1, 'min_samples_split': 10, 'n_estimators': 150}
  OOB Score = 0.8089
  Train Accuracy = 87.65%
  Validation Accuracy = 80.55%
  Test Accuracy = 80.84%

- **Scoring Function - OOB Error**
  Optimal Parameters - {'max_features': 0.1, 'min_samples_split': 10, 'n_estimators': 450}
  OOB Score = 0.8103
  Train Accuracy = 87.59%
  Validation Accuracy = 80.62%
  Test Accuracy = 80.75%

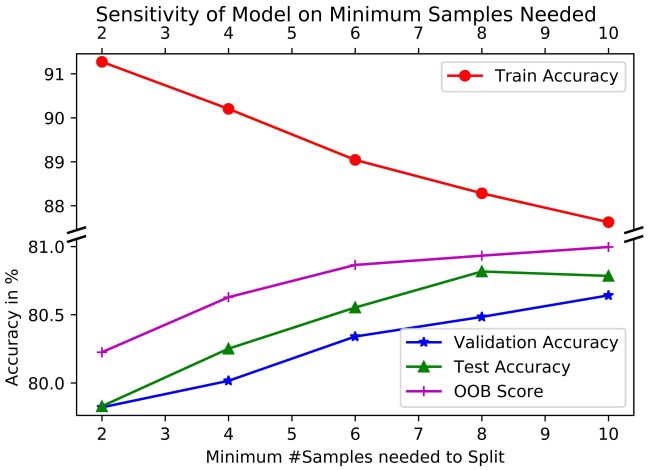
These numbers are within 1-2% of the numbers that we reported for Post-Pruned Trees. The Validation Accuracy achieved by the best tree was 79.04% and the Test Accuracy was 79.06% with 6821 nodes. We see that the Post Pruned trees are able to achieve almost the same accuracy as Random Forest overcoming over-fitting.



(a) Sensitivity of the Model on Number of Estimators



(b) Sensitivity of the Model on the number of maximum features



(c) Sensitivity of the Model on the minimum number of samples needed for split

Figure 4: Accuracy trends with changing parameters in the Random Forest

2.4 Part D: Sensitivity Analysis of Parameters on the Optimal Random Forest

Sensitivity Analysis is done to check how dependent the predictions of a Decision Tree or Random Forest, in this case, are on the different input parameters. We study the sensitivity of the optimal random forest classifier that was found in the grid search on the parameters studied in the

previous section.

Figure 4a shows how the accuracy of the forest classifier changes by changing the number of estimators in the forest by keeping the other parameters at their optimal value. Since we have selected the model based on the OOB error, we have also plotted the OOB score in the graph for comparison. The accuracy of the model does not change much and the variation is between $\pm 0.2\%$. But the maximum OOB score is obtained at n_estimators = 450 which is the optimal value found.

Figure 4b shows how the accuracy of the model changes by changing the maximum number of attributes that would be used to train the different trees in the forest. The figure shows that the model is a bit sensitive to this parameter since there is a steep drop in the accuracy by 0.4% as the number of attributes increase. The highest accuracy and the highest OOB score is obtained at max_features = 0.1 which is around 50 features as found.

Figure 4c shows how the accuracy of the model changes by changing the minimum number of samples needed at a node to split further. We see that model is a lot sensitive to this parameter and we see a change in the accuracy by $\pm 0.7 - 0.8\%$ which is high. The highest OOB score and the highest accuracy is achieved at min_samples_split=10.

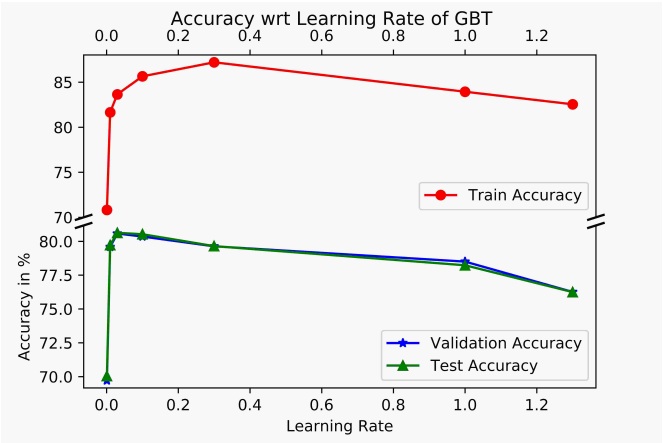
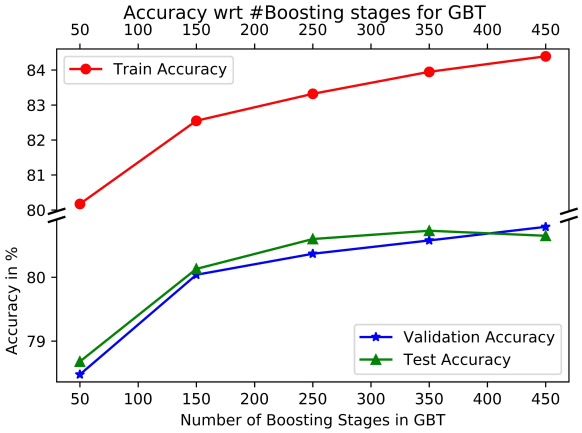


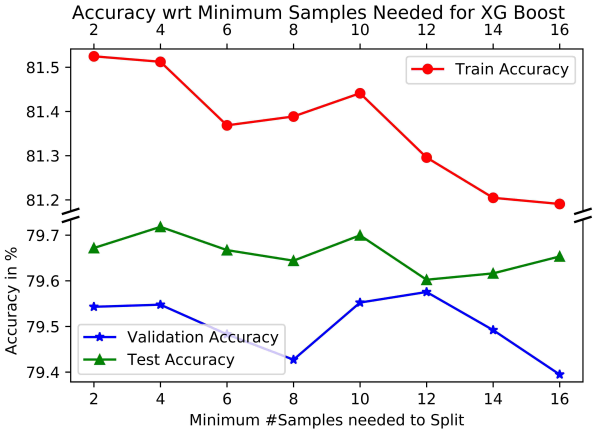
Figure 5: Accuracy comparison of Gradient Boosted trees with different learning rates

2.5 Gradient Boosted Trees

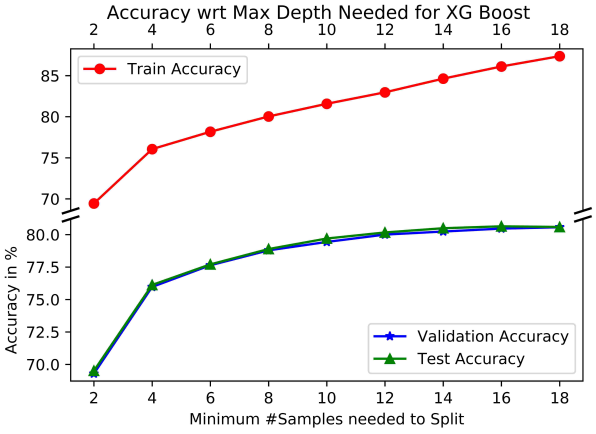
Boosting is a technique of successively building stronger models from weaker models by refocusing on the examples that were incorrectly predicted by the previous model. In Gradient Boosting algorithm, weak learners are added in an additive stage-wise fashion while minimizing an arbitrary loss function of the model in a gradient descent like fashion. A learning rate is used to decide the descent. Figure 5 shows the behaviour of the Gradient Boosted Models with different Learning rates. We see that the model achieves high accuracy at smaller learning rates of 0.01, 0.1 but the accuracy starts to degrade at higher learning rates.



(a) Accuracy trend with increasing number of estimators



(b) Accuracy trend with different number of minimum samples needed to split at node



(c) Accuracy with different depth of trees (Default for GBT is depth = 3)

Figure 6: Accuracy comparison of Gradient Boosted trees with different model parameters

The Gradient Boosting algorithm uses a loss function to optimize which mostly depends on the problem to be solved. We have used the Deviance Loss function which is equivalent to the logistic regression cost function. The steps of adding weak learners are taken according to the learning rate.

The Gradient Boosting Algorithm also uses a weak learner which is mostly a decision tree with constrained height or number of nodes or number of leaf nodes. Figure 6c shows how the GB trees behave with increasing

depth of the weak learners added to the model. We see that upto a certain depth of the tree, the accuracy increase and then plateaus. This is because the learners have to be weak. So, even very small depth trees learn very well in gradient boosted methods.

Figure 6a shows how the algorithm performs as the number of boosting stages are increased. The graph shows that the algorithm is fairly resistant to overfitting and even on increasing the number of boosting stages to 450 we keep seeing better and better performance of the model.

Figure 6b shows how the model performs by adding a restriction of minimum samples needed to split while learning a weak learner. The figure shows the model is highly sensitive to this constraint on the leaner.

The accuracy obtained using Gradient Boosted Trees is almost similar to what we obtained using Random Forest on this dataset. But since it is a stage-wise algorithm, it takes more time to build as compared to random forest where the trees are all grown in parallel. The random forest is almost 5x faster than Gradient Boosted Trees.

3 Conclusions

This report is a summary of the third assignment, first part, done for Machine Learning course. We obtain a good understanding of how the algorithms that we talked about in this report are implemented and we also analyzed the affect of different hyper-parameters on the learning.