# CptS 570 Machine Learning, Fall 2018
# Homework #3

Due Date: Tue, Nov 13 (9:10am)

NOTE 1: Please use a word processing software (e.g., Microsoft word or Latex) to write your answers and submit a printed copy to me at the beginning of class on Oct 23. The rationale is that it is sometimes hard to read and understand the hand-written answers.

NOTE 2: Please ensure that all the graphs are appropriately labeled (x-axis, y-axis, and each curve). The caption or heading of each graph should be informative and self-contained.

1. (**15 points**) We need to perform statistical tests to compare the performance of two learning algorithms on a given learning task. Please read the following paper and briefly summarize the key ideas as you understood:

   Thomas G. Dietterich: Approximate Statistical Test For Comparing Supervised Classification Learning Algorithms. Neural Computation 10(7): 1895-1923 (1998) `http://sci2s.ugr.es/keel/pdf/algorithm/articulo/dietterich1998.pdf`

2. (**5 points**) Please read the following paper and briefly summarize the key ideas as you understood:

   Thomas G. Dietterich (1995) Overfitting and under-computing in machine learning. Computing Surveys, 27(3), 326-327.
   `http://www.cs.orst.edu/~tgd/publications/cs95.ps.gz`

3. (**10 points**) Please read the following paper and briefly summarize the key ideas as you understood:

   Thomas G. Dietterich (2000). Ensemble Methods in Machine Learning. J. Kittler and F. Roli (Ed.) First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science (pp. 1-15). New York: Springer Verlag.
   `http://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf`

4. (**10 points**) Please read the first five sections of the following paper and briefly summarize the key ideas as you understood:

   Jerome Friedman (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5), pp 1189–1232.
   `https://statweb.stanford.edu/~jhf/ftp/trebst.pdf`

5. (**10 points**) Please read the following paper and briefly summarize the key ideas as you understood:

   Tianqi Chen, Carlos Guestrin: XGBoost: A Scalable Tree Boosting System. KDD 2016.
   `https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf`

6. (**25 points**) Empirical analyis question. Income Classifier using Bagging and Boosting. You will use the *Adult Income* dataset from HW1 for this question. You can use Weka or scikit-learn software.

   a. Bagging (weka.classifiers.meta.Bagging). You will use decision tree as the base supervised learner. Try trees of different depth (1, 2, 3, 5, 10) and different sizes of bag or ensemble, i.e., number of trees (10, 20, 40, 60, 80, 100). Compute the training accuracy, validation accuracy, and testing accuracy for different combinations of tree depth and number of trees; and plot them. List your observations.

b. Boosting (weka.classifiers.meta.AdaBoostM1). You will use decision tree as the base supervised learner. Try trees of different depth (1, 2, 3) and different number of boosting iterations (10, 20, 40, 60, 80, 100). Compute the training accuracy, validation accuracy, and testing accuracy for different combinations of tree depth and number of boosting iterations; and plot them. List your observations.

7. (**25 points**) Automatic hyper-parameter tuning via Bayesian Optimization. For this homework, you need to use BO software to perform hyper-parameter search for Bagging and Boosting classifiers: two hyper-parameters (size of ensemble and depth of decision tree).

   You will employ Bayesian Optimization (BO) software to automate the search for the best hyper-parameters by running it for 50 iterations. Plot the number of BO iterations on x-axis and performance of the best hyper-parameters at any point of time (performance of the corresponding trained classifier on the validation data) on y-axis.

   Additionally, list the sequence of candidate hyper-parameters that were selected along the BO iterations.

   You can use one of the following BO softwares or others as needed.
   Spearmint: https://github.com/JasperSnoek/spearmint
   SMAC: http://www.cs.ubc.ca/labs/beta/Projects/SMAC/

**Instructions for Code Submission and Output Format.**

Please follow the below instructions. It will help us in grading your programming part of the homework. We will provide a dropbox folder link for code submission.

- Supported programming languages: Python, Java, C++
- Store all the relevant files in a folder and submit the corresponding zipfile named after your student-id, e.g., 114513209.zip
- This folder should have a script file named

  ```
  run_code.sh
  ```

  Executing this script should do all the necessary steps required for executing the code including compiling, linking, and execution
- Assume relative file paths in your code. Some examples:

  ```
  ‘‘./filename.txt’’ or ‘‘../hw2/filename.txt’’
  ```
- The output of your program should be dumped in a file named “output.txt”
- Make sure the output.txt file is dumped when you execute the script

  ```
  run_code.sh
  ```
- Zip the entire folder and submit it as

  ```
  <student_id>.zip
  ```

## Grading Rubric

Each question in the students work will be assigned a letter grade of either A,B,C,D, or F by the Instructor and TAs. This five-point (discrete) scale is described as follows:

- **A) Exemplary (=100%).**
  Solution presented solves the problem stated correctly and meets all requirements of the problem.
  Solution is clearly presented.
  Assumptions made are reasonable and are explicitly stated in the solution.
  Solution represents an elegant and effective way to solve the problem and is not overly complicated than is necessary.

- **B) Capable (=75%).**
  Solution is mostly correct, satisfying most of the above criteria under the exemplary category, but contains some minor pitfalls, errors/flaws or limitations.

- **C) Needs Improvement (=50%).**
  Solution demonstrates a viable approach toward solving the problem but contains some major pitfalls, errors/flaws or limitations.

- **D) Unsatisfactory (=25%)**
  Critical elements of the solution are missing or significantly flawed.
  Solution does not demonstrate sufficient understanding of the problem and/or any reasonable directions to solve the problem.

- **F) Not attempted (=0%)**
  No solution provided.

The points on a given homework question will be equal to the percentage assigned (given by the letter grades shown above) multiplied by the maximum number of possible points worth for that question. For example, if a question is worth 6 points and the answer is awarded a $B$ grade, then that implies 4.5 points out of 6.

# Assignment 3

Sheryl Mathew (11627236)

November 13, 2018

1. **Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms**

Designing and evaluating Statistical tests involves 4 different sources of variation that must be controlled by each test.

- Random variation in the selection of test data: 2 classifiers perform the same on the entire dataset but on the randomly selected test data one might perform better than the other.
- Random variation in the selection of training data: 2 classifiers have same accuracy on the entire dataset but on the randomly selected training data one might perform better than the other. Also, any changes in the training data which occurs when we add or remove data points will cause drastic change in the classifier produced by the learning algorithm. This behavior is called instability.
- Internal randomness in the learning algorithm: Algorithms produce different hypothesis when random starting state of the algorithm changes.
- Random classification error: If a fixed fraction $\eta$ of the test data points is mislabeled then no learning algorithm should achieve an error rate less than $\eta$.

*McNemar's test:*

Procedure:

Sample data S in divided into training set R and test set T. Perform algorithms A and B on R to get classifiers $F_A$ and $F_B$.

Analysis:

The test is based on $X^2$ statistic test which compares the distribution of counts expected under null hypothesis to observed counts to get the goodness-of-fit. If the null hypothesis is correct then we reject the null hypothesis which says $F_A$ and $F_B$ should have the same error rate. Instead we consider the hypothesis that $F_A$ and $F_B$ have different performances when trained on R.

Observations:

- It should be applied when we know that the source of variations is small since only a single training set is considered by the algorithm.
- An assumption is made that the relative difference of the training set of size |R| and training set of size |S| is the same since we only compute the algorithm performance on |R| instead of |S|

Equation:

$$X^2 = \frac{(|\eta_{01} - \eta_{10}| - 1)^2}{\eta_{01} + \eta_{10}}$$

$\eta_{01}$: number of examples misclassified by $F_A$ and not by $F_B$
$\eta_{10}$: number of examples misclassified by $F_B$ and not by $F_A$

*Test for difference of two proportions:*

Procedure:

Sample data S in divided into training set R and test set T. Perform algorithms A and B on R to get classifiers $F_A$ and $F_B$. Find the error rate $P_A$ and $P_B$ of A and B which involves the proportion of examples the classifier incorrectly classifies on T.

Analysis:

The number of misclassifications is considered as a binomial distribution. An assumption is made that $P_A$ and $P_B$ are independent which leads to the conclusion that $P_A$ - $P_B$ is normally distributed. We can reject the null hypothesis when the z statistic, $|z| > 1.96$.

Issues:

- $P_A$ and $P_B$ are not independent since they are measured on the same T.
- Does not measure variation due to training set choice or internal variation of algorithm
- Does not measure the performance on the training set of size |S| but on training set of size |R|

Observations:

- The lack of independence is correcting by rectifying the standard deviation equation and the resulting z-statistic is the square root of $X^2$ statistic test.
- For small sample size, the accuracy is the same for both McNemar's test and Test for difference of two proportions.

Equation:

$$z = \frac{|\eta_{01} - \eta_{10}| - 1}{\sqrt{\eta_{01} + \eta_{10}}}$$

$\eta_{01}$: number of examples misclassified by $F_A$ and not by $F_B$
$\eta_{10}$: number of examples misclassified by $F_B$ and not by $F_A$

*Resampled paired t test:*

Procedure:

Consider 30 trials. In each trial, sample data S in divided into training set R and test set T. Perform algorithms A and B on R to get classifiers $F_A$ and $F_B$. Find the error rate $P_A^i$ and $P_B^i$ of A and B for trial $i$ which involves the proportion of examples the classifier incorrectly classifies on T.

Analysis:

The number of misclassifications is considered as a binomial distribution. An assumption is made that $P_A$ and $P_B$ are independent which leads to the conclusion that $P = P_A$ - $P_B$ is normally distributed. 30 such differences are drawn to calculate the Student's t test. We can reject the null hypothesis when the t statistic, $|t| > 2.045$.

Issues:

- $P_A^i$ and $P_B^i$ are not independent therefore the individual differences will not be normally distributed.
- $P^i$'s are not independent since the test and training sets in the trials overlap.

Equation:

$$t = \frac{\bar{P}\sqrt{n}}{\sqrt{\frac{\sum_{i=1}^{n}(P^i - \bar{P})^2}{n-1}}}$$

$n$: number of examples

$\bar{P} = \frac{1}{n}\sum_{i=1}^{n} P^i$

*k-fold cross-validated paired t test:*

Procedure:

Sample data S in divided into k disjoint sets of equal size $T_1, T_2, \ldots, T_k$. k trials are conducted. In each trial, the test set is $T_i$ and training set is the union of all of the other $T_j, j \neq i$ . Perform algorithms A and B on R to get classifiers $F_A$ and $F_B$. Find the error rate $P_A^i$ and $P_B^i$ of A and B for trial $i$ which involves the proportion of examples the classifier incorrectly classifies on T.

Analysis:

Calculate the Student's t test for the above data received.

Issues:

- Training sets overlap which prevents the statistical test from obtaining a good estimate of the amount of variation that can be observed.
- Performance of the cross validation is correlated instead of being independent.

Observations:

- Each test set is independent of others

*5X2 cv paired t test:*

Procedure:

Perform 5 replications of 2-fold cross validation. In each replication, the data is randomly partitioned into 2 equal sized sets $S_1$ and $S_2$. Algorithms A and B are trained on each set and tested on the other set.

Analysis:

4 error estimates are calculated. $P_A^1$ and $P_B^1$ (train: $S_1$, test: $S_2$). $P_A^2$ and $P_B^2$ (train: $S_2$, test: $S_1$). This leads to 2 estimated difference $P^1$ and $P^2$. From these estimates the variance s is calculated. After which t-statistic is derived based on the below assumptions:

- Normal approximation is applied to binomial distribution
- Pairwise independent of $P_i^1$ and $P_i^2$ for all $i$
- Assume independence between all the $s_i$ values
- Assume independence between numerator and denominator of t-statistic test

Equation:

$$t = \frac{\sqrt{n}\,(\bar{X} - \mu)}{\sqrt{\frac{S^2}{n-1}}}$$

$n$: number of examples

$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$ : sample mean

$S^2 = \sum_{i=1}^{n} (X_i - \bar{X})^2$ : sum of squared deviation from the mean

Conclusions**:**

- Type I error rate gives the confidence that an observed performance difference is real. 10-fold cross validated t test has Type I error which exceeds 0.05. All other tests have acceptable Type I error rates.
- Power of a test tells whether there is a difference between two learning algorithms. Cross-validated test has the highest power curve.
- 5X2 cv test is the most powerful since it has an acceptable Type I error rate and a good power curve.

## 2. Overfitting and under-computing in machine learning

Learning algorithms search the hypothesis class for a function that fits the given data. The search for the function is done by defining an objective function over which several algorithms are run to find that function which will minimize the objective function. The learning algorithm is trained on a set of training data and then used to perform predictions on new data points. The goal is to maximize predictive accuracy over the new data.

Overfitting occurs when we try to fit the training data in such a way that even the noise in the data is fitted. This is done by the algorithm by memorizing peculiarities of the training data instead of finding a general predictive rule. This leads to the objective function being incorrect. This is rectified by adding penalty terms to the objective function to predict the off-training-set accuracy from on-training-set accuracy. The penalty terms include regularization methods, generalized cross validation, minimum-description-length among others. Optimizing the objective function with penalty terms is computationally complex.

Simple gradient descent and greedy algorithms are considered to embody the correction terms. This can be explained based on decision trees. Instead of trying to solve the objective function of finding the smallest tree that fits the data and then fitting the penalty terms and perform the function again, we consider the optimal approach to be finding the objective function which finds a tree that minimizes some combination of decision tree size plus a penalty term that corrects the difference between training and test data.

Under-computing is when it is optimal to be suboptimal. The original optimization problem was intractable so we solve another problem using a polynomial-time algorithm. This avoids overfitting as shown in the above decision tree example.

## 3. Ensemble Methods in Machine Learning

Ensemble of classifiers is a set of classifiers whose individual decisions are combined either by weighted or unweighted voting to classify new examples. Ensembles are usually more accurate than their individual classifier. For this to be satisfied, the classifiers must be accurate and diverse. An accurate classifier is one which has an error rate better than randomly guessing the values for new x. Two classifiers are said to be diverse if they make different errors on new data points.

Fundamental Reasons to construct good ensembles:

*Statistical*: When the training data is too small when compared to the size of the hypothesis space in which we are searching for the best hypothesis, the learning algorithm will find different hypotheses that all give the same accuracy. Instead if we construct an ensemble of all the accurate classifiers, then the algorithm can average their votes and reduce the risk of choosing the wrong classifier.

*Computational*: An ensemble constructed by running local search for many different staring points for algorithms like decision tree and deep neural networks provides a better approximation to the true unknown function than any of the individual classifier since the optimal training for them in NP-hard.

*Representational*: For an individual classifier, the true function f cannot be represented by any of the hypotheses in H. In such cases, an ensemble can be used by forming weighted sums of the hypotheses drawn from H which expands the space of representable functions.

Methods for Constructing Ensembles:

*Bayesian Voting*: This is an ensemble method in which the ensemble consists of all of the hypotheses in H, each weighted by its posterior probability P(h|S).

$$P(f(x) = y|S, x) = \sum_{h \epsilon H} h(x)P(h|S)$$

When training sample is small, many hypotheses will have large posterior probabilities and the voting process averages these to marginalize away the uncertainty of f. When training sample is large, only one hypothesis will have substantial posterior probability and the ensemble shrinks to contain only one hypothesis.

*Manipulating training examples*: The ensembles can be constructed by manipulating the training examples to generate multiple hypotheses. This is done by running the learning algorithm multiple times and each time giving the input a different subset of the training examples.

*Manipulating input features*: When the input features are highly redundant, the input feature subset can be selected by different techniques to develop a highly efficient ensemble classifier.

*Manipulating output targets*: When the number of classes K is large, then error-correcting output coding is performed to the output variable. This is done by randomly partitioning the K classes into 2 subsets A and B. The input data is relabeled such that any of the original classes in set A is 0 ang original classes in set B is 1. This relabeled data is given to the learning algorithm which constructs a classifier h. By repeating the process L times with different subsets of A and B, we get an ensemble of classifiers $\{h_1, h_2 \dots h_L\}$. For a new data point x, each h will classify x and the class with the highest number of votes will be the prediction of the ensemble.

*Injecting Randomness*: Randomness can be added in different ways. If an algorithm is applied to same training examples but with different initial weights, each classifier will be different from the other. Noise can be injected into the input features for the learning algorithm. While constructing Bayesian ensemble, randomness is introduced to the learning process. But instead of taking uniform vote, each hypothesis receives vote proportional to its posterior probability.

Shortcomings of Decision Tree Algorithm:

*Statistical Support*: If a true function f is represented by a small decision tree, no ensemble needed. If f is represented by a large decision tree then a large training set is required to fit the tree leading to a statistical problem.

*Computation*: A decision tree makes series of decisions greedily while building the tree. If one of the decisions is incorrect, then the training data will be incorrectly partitioned and hence all subsequence decisions will be affected.

*Representation*: Decision tree's decision boundaries are represented as rectangular partitions of the input space. When the decision boundaries are not orthogonal to the coordinate axes then infinite size is required to represent the boundaries correctly.

Results for Ensemble performance with Decision tree as base learner:

- When datasets had little or no noise, AdaBoost gives the best results. Bagging and Randomized trees have similar performance. But on very large datasets, Randomized trees perform better.
- When 20% artificial noise is added to the datasets, Bagging gives the best results. AdaBoost overfits the data badly. Randomized trees do not perform very well.

Explanation of the Results for Ensemble performance with Decision tree as base learner:

*AdaBoost*: In AdaBoost, each new decision tree, is constructed to eliminate the residual errors that have not been properly handles by the weighted vote of the previously constructed tree. It is trying to optimize the weighted vote. Directly optimizing an ensemble increases the risk of overfitting. Therefore, AdaBoost gives good performance in low noise cases since it is able to optimize the ensemble without overfitting. But in high noise cases, AdaBoost puts a large amount of weight on the mislabeled examples which leads it to overfit very badly.

*Bagging*: In bagging, each decision tree is built independently of the others by manipulating the input data. It does well in both noisy and noise-free cases since it focuses on the statistical problem.

*Randomized trees*: In randomized trees, each decision tree is built independently of the others by directly altering the choices of C4.5. It does well in noise-free cases and not that well on noisy and since it focuses on the statistical problem and noise increases this statistical problem. It performs well on very large datasets since bootstrap replicates of large training set is similar to the training set itself. It creates diversity under all conditions at the risk of generating low-quality decision tree.

## 4. Greedy function approximation: A gradient boosting machine

Function estimation consists of a random output variable y and set of input variables $x = \{x_1, x_2, \dots x_n\}$. The training sample is used to obtain an estimate F(x) of the function $F^*(x)$ that minimizes the expected value of a loss function L(y,F(x)). F(x) is restricted to be a member of a parameterized class of functions F(x;P) where $P = \{P_1, P_2 \dots\}$ is a finite set of parameters.

$$F(x;\{\beta_m, a_m\}_1^M) = \sum_{m=1}^{M} \beta_m h(x;a_m)$$

h(x;a): parameterized function of the input variables x characterized by parameters $a = \{a_1, a_2 \dots a_n\}$
Here each of the functions h(x;a) is a small regression tree. For the regression tree, $a_m$ are the splitting variables, split locations, terminal node means of the individual trees.

Choosing a parameterized model F(x;P) converts function optimization problem to parameter optimization problem. This is done by using Numerical optimization methods to solve $P^*$.

$$P^* = \sum_{m=0}^{M} P_m$$

$P_0$ is the initial guess and $\{P_m\}_1^M$ is the successive boosts based on the sequence of preceding steps. Steepest-descent is the frequently used numerical minimization methods. When we apply numerical optimization in function space there are infinite number of parameters to be considered. But in datasets only a finite number of $\{F(x_i)\}_1^N$ exists with respect to function F(x)

To minimize the data based estimate of expected loss, greedy-stagewise approach is used. The below 2 equations are called boosting.

$$(\beta_m, a_m) = \arg\min \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i;a))$$

$$F_m(x) = F_{m-1}(x) + \beta_m h(x;a_m)$$

L(y,F) : squared-error loss
$\{h(x;a_m)\}_1^M$ : basis functions
$h(x$;a) : base learner

If the loss function or base learner is difficult to compute, then give approximator $F_{m-1}(x)$, $\beta_m h(x;a_m)$ is the greedy step towards the data based estimate of $F^*(x)$ under the constraint that $h(x;a_m)$ must be a member of $h(x$;a). This is the steepest-descent step. The constraint is applied to the unconstrained solution by fitting $h(x$;a) to the pseudo-responses. This converts the function minimization problem to least-squares function minimization problem.

When gradient boosting is performed on squared-error loss, it produces the usual stagewise approach of iteratively fitting current residuals. This is called least-squares boosting. When regression trees are the base learner, least-squares are used to induce tress since squares-error loss is more quickly updated than mean-absolute-deviation when searching for splits while building the tree. This is called least-absolute-deviation regression.

M-Regression techniques are used to attempt resistance to long-tailed error distributions and outliers as well as maintaining high efficiency for normally distributed errors. The algorithm for boosting regression trees based on Huber loss is similar to least-squares boosting for normally distributed errors, similar to least-absolute-deviation regression with very long tailed distributions and superior to both for error distributions with only moderately long tails.

In prediction problems, when we fit the training data too closely leads to overfitting. This causes the expected loss to stop decreasing and start increasing. This is prevented by implementing Regularization methods. Regularization methods do this by constraining the fitting procedure which here is the number of components M. By controlling the value of M, we can regulate the degree to which the expected loss on the training data can be minimized. M can be chosen using cross-validation.

## 5. XGBoost: A Scalable Tree Boosting System

XGBoost is a scalable machine learning system for tree boosting. The scalability is due to:

- novel tree learning algorithm for handling sparse data
- weighted quantile sketch procedure enables handling instance weights in approximate tree learning.
- parallel and distributed computing enables quicker model exploration
- out-of-core computation allows the processing of millions of examples on desktop

Algorithms:

*Approximate and Exact Greedy tree learning algorithm*: In tree learning, exact greedy algorithm is used to find the best split by enumerating over all the possible splits on all the features greedily. But it is difficult to this efficiently when the data does not fit into the memory. This is overcome using approximate algorithm with both local and global proposal methods.

*Weighted quantile sketch algorithm*: To solve the problem that there is no existing quantile sketch for weighted datasets, a distributed weighted quantile sketch algorithm is developed. It handles weighted data with a provable theoretical guarantee and includes a data structure which supports merge and prune operations as well as maintaining a certain accuracy level.

*Sparsity aware algorithm*: The input data is sparse and the sparsity pattern should be known to the algorithm. This is accomplished by adding a default direction in each tree node. So, whenever a value is missing in the input, the instance is classified in the default direction. Each branch has 2 choices of default direction. The optimal default direction is learnt from the data. Only the non-missing entries are visited. All the sparsity patterns are handled in a unified manner. The computational complexity is linear to number of non-missing entries in the input.

System Design:

*Column block for parallel learning*: The data is stored in in-memory units called blocks. Data in each block is stored in CSC (compressed column) format where each column is sorted by the corresponding feature value. This input data layout is computed only once before training and reused later. For exact greedy algorithms, the entire dataset is stored in a single block and the split search algorithm is run by linearly scanning over the pre-sorted entries. For approximate algorithms, multiple blocks are used with each block corresponding to a subset of rows in the dataset. Column block structure supports column subsampling. Collecting statistics for each column is parallelized.

*Cache-aware Access*: For exact greedy algorithms, an internal buffer is allocated for each thread. The gradient statistics are fetched into it and then accumulation is performed in mini-batch. This reduces the runtime overhead when the number of rows is large. For approximate algorithms, the maximum number of examples that can be stored in a block is called block size. $2^{16}$ is the optimum block size which balances the cache property and parallelization.

*Blocks for out-of-core computation*: The data is divided into multiple blocks and each block is stored on disk. Each block is compressed by columns and decompressed during computation by an independent thread when loading into main memory (Block compression). The data is sharded onto multiple disks and the pre-fetcher which is assigned to each disk is used to fetch the data into an in-memory buffer and the training thread reads the data from each buffer (Block Sharding). This increases the throughput of disk reading.

Results:

*Classification*: XGBoost and scikit-learn show a better classification performance when compared to R's GBM with XGBoost running 10X faster than scikit-learn.

*Learning to Rank*: XGBoost runs faster when compared to pGBRT. In XGBoost, subsampling columns reduces running time and gives higher performance since subsampling prevents overfitting.

*Out-of-core Experiment*: Process 1.7 billion examples on a single machine

*Distributed Experiment*: XGBoost runs faster than Spark MLLib and H2O by using out-of-core computing and smoothly scaling to all 1.7 billion examples with limited computing resources. XGBoost's performance improves linearly as we add more machines. XGBoost was able to handle the entire 1.7 billion data with only 4 machines.

## 6. Bagging and Boosting

Bagging:

Train Accuracies for Number Decision Trees vs Depth of Tree: Bagging

| | 10 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| 1 | 75.160 | 75.160 | 75.160 | 75.160 | 75.160 | 75.160 |
| 2 | 78.372 | 78.372 | 78.372 | 78.372 | 78.372 | 78.372 |
| 3 | 81.548 | 81.384 | 81.364 | 80.444 | 80.508 | 81.340 |
| 5 | 82.632 | 82.280 | 82.212 | 82.336 | 82.352 | 82.328 |
| 10 | 85.572 | 85.528 | 85.732 | 85.752 | 85.644 | 85.760 |

Development Accuracies for Number Decision Trees vs Depth of Tree: Bagging

| | 10 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| 1 | 76.061008 | 76.061008 | 76.061008 | 76.061008 | 76.061008 | 76.061008 |
| 2 | 78.978780 | 78.978780 | 78.978780 | 78.978780 | 78.978780 | 78.978780 |
| 3 | 82.095491 | 82.095491 | 82.095491 | 81.962865 | 82.095491 | 81.233422 |
| 5 | 83.090186 | 83.156499 | 83.090186 | 82.891247 | 82.824934 | 82.824934 |
| 10 | 84.084881 | 83.819629 | 83.753316 | 84.283820 | 83.885942 | 83.885942 |

Testing Accuracies for Number Decision Trees vs Depth of Tree: Bagging

| | 10 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| 1 | 73.939394 | 73.939394 | 73.939394 | 73.939394 | 73.939394 | 73.939394 |
| 2 | 76.410256 | 76.410256 | 76.410256 | 76.410256 | 76.410256 | 76.410256 |
| 3 | 79.860140 | 79.067599 | 79.860140 | 79.067599 | 79.114219 | 79.067599 |
| 5 | 81.771562 | 81.724942 | 81.818182 | 81.724942 | 81.724942 | 81.678322 |
| 10 | 81.864802 | 82.004662 | 82.191142 | 82.097902 | 82.097902 | 81.864802 |

Train Accuracies for Depth of Tree vs Number of Decision Trees: Bagging

| | 1 | 2 | 3 | 5 | 10 |
|---|---|---|---|---|---|
| 10 | 75.16 | 78.372 | 81.312 | 82.236 | 85.472 |
| 20 | 75.16 | 78.372 | 80.524 | 82.572 | 85.584 |
| 40 | 75.16 | 78.372 | 80.528 | 82.548 | 85.656 |
| 60 | 75.16 | 78.372 | 81.372 | 82.284 | 85.780 |
| 80 | 75.16 | 78.372 | 81.360 | 82.404 | 85.672 |
| 100 | 75.16 | 78.372 | 80.512 | 82.476 | 85.780 |

Development Accuracies for Depth of Tree vs Number of Decision Trees: Bagging

| | 1 | 2 | 3 | 5 | 10 |
|---|---|---|---|---|---|
| 10 | 76.061008 | 78.97878 | 82.029178 | 83.090186 | 83.355438 |
| 20 | 76.061008 | 78.97878 | 81.299735 | 82.758621 | 84.084881 |
| 40 | 76.061008 | 78.97878 | 82.095491 | 82.891247 | 83.952255 |
| 60 | 76.061008 | 78.97878 | 81.366048 | 83.090186 | 83.952255 |
| 80 | 76.061008 | 78.97878 | 81.366048 | 82.824934 | 84.084881 |
| 100 | 76.061008 | 78.97878 | 81.366048 | 82.957560 | 83.952255 |

Testing Accuracies for Depth of Tree vs Number of Decision Trees: Bagging

| | 1 | 2 | 3 | 5 | 10 |
|---|---|---|---|---|---|
| 10 | 73.939394 | 76.410256 | 79.067599 | 81.678322 | 82.097902 |
| 20 | 73.939394 | 76.410256 | 79.720280 | 81.818182 | 81.724942 |
| 40 | 73.939394 | 76.410256 | 79.114219 | 81.818182 | 82.191142 |
| 60 | 73.939394 | 76.410256 | 79.114219 | 81.771562 | 82.284382 |
| 80 | 73.939394 | 76.410256 | 79.114219 | 81.631702 | 81.911422 |
| 100 | 73.939394 | 76.410256 | 79.114219 | 81.771562 | 81.958042 |

Observations:

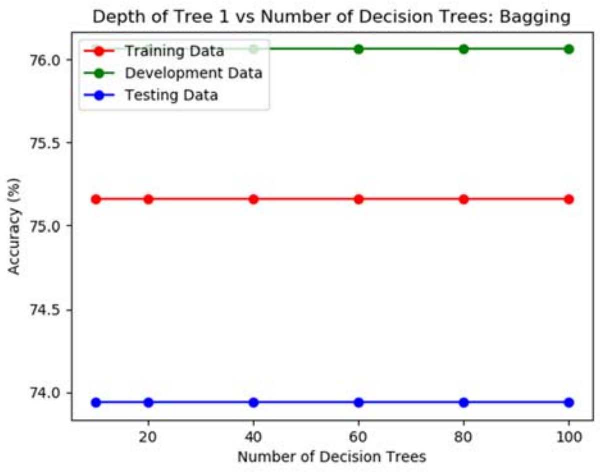
Number of Decision Trees vs Depth of Tree

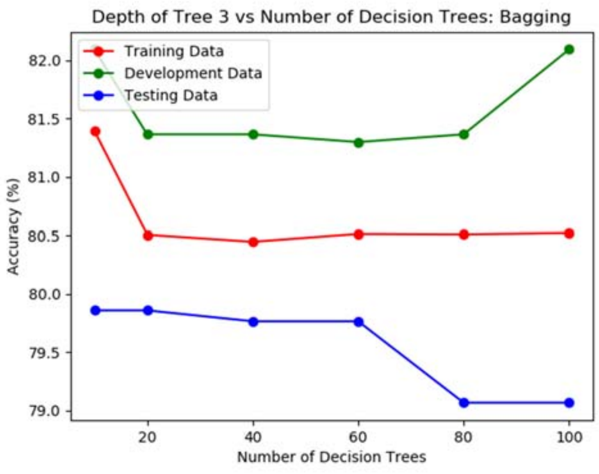
- For training data, when the number of decision tree increases the accuracy remains almost same.
- For development data, for depths 3 and 5, when the number of decision tree increases the accuracy increases.
- For testing data, when the number of decision tree increases the accuracy remains almost same.

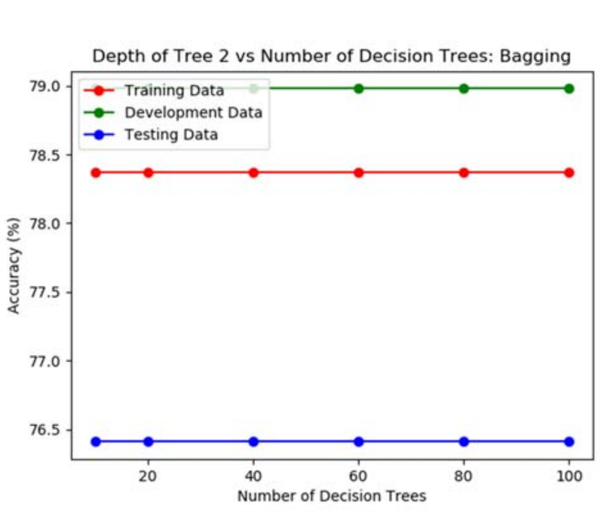
Depth of Tree vs Number of Decision Trees

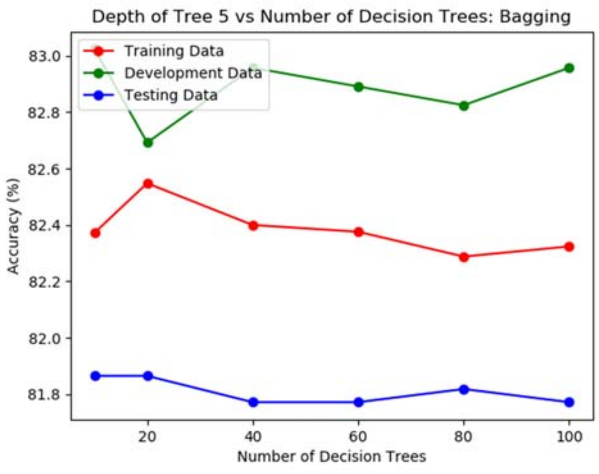
- For training data, when the depth increases the accuracy increases.
- For development data, when the depth increases the accuracy increases.
- For testing data, when the depth increases the accuracy increases.

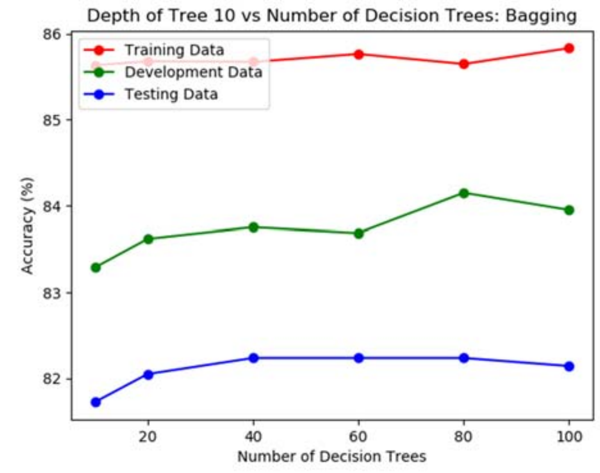
## Number of Decision Trees vs Depth of Tree



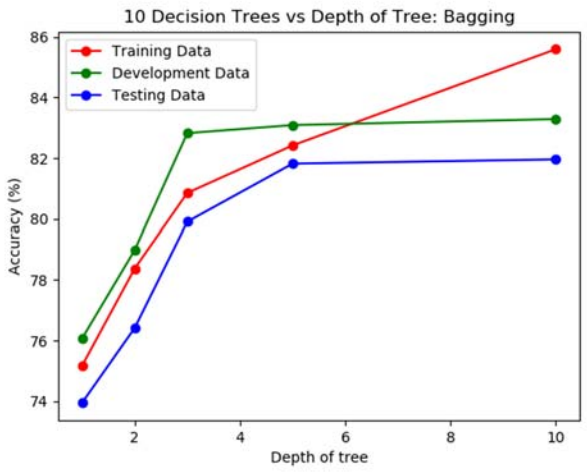


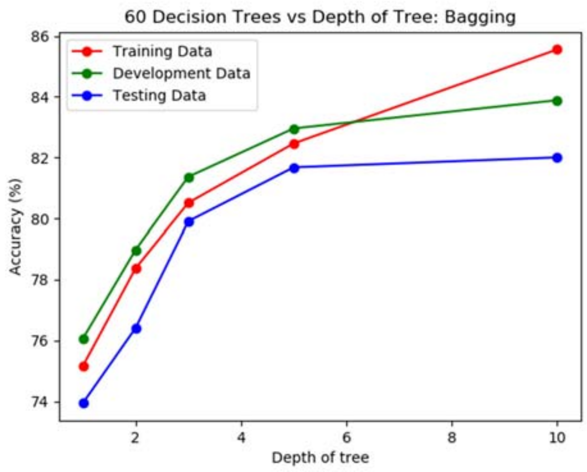


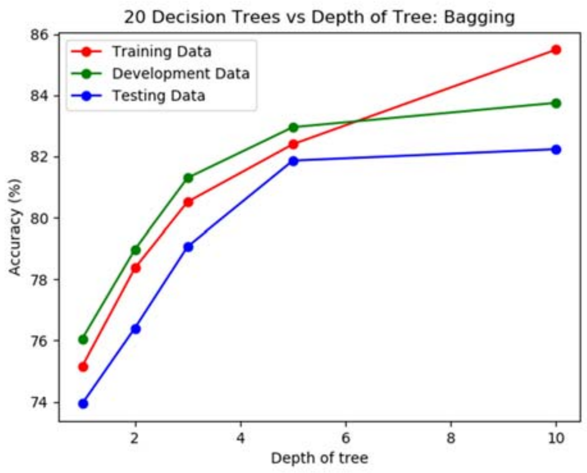


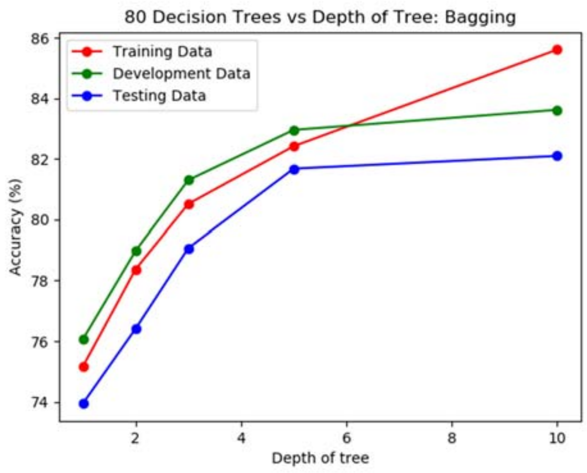


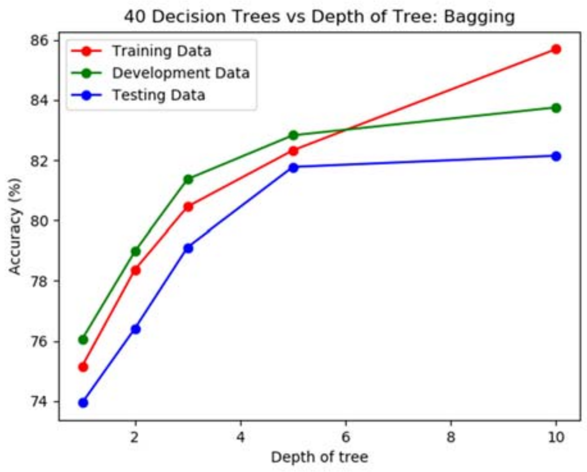
Depth of Tree vs Number of Decision Trees

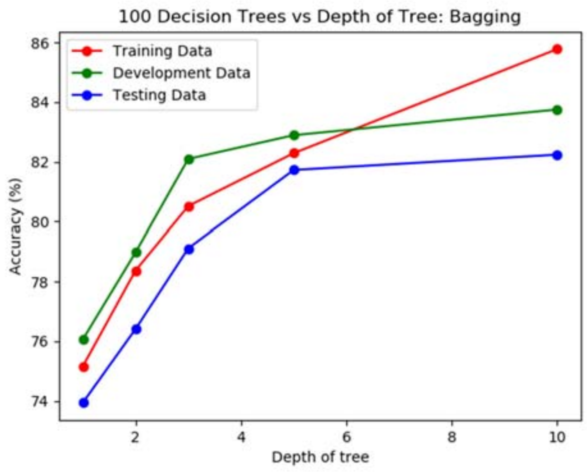












Boosting:

Train Accuracies for Number of boosting iterations vs Depth of Tree: Boosting

| | 10 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| 1 | 81.280 | 82.876 | 83.436 | 83.524 | 83.548 | 83.556 |
| 2 | 82.976 | 83.568 | 83.876 | 84.144 | 84.188 | 84.436 |
| 3 | 83.252 | 83.572 | 84.280 | 84.516 | 84.836 | 85.192 |
| 5 | 84.260 | 84.720 | 85.760 | 86.644 | 87.460 | 88.144 |
| 10 | 88.072 | 91.112 | 95.444 | 96.712 | 96.716 | 96.716 |

Development Accuracies for Number of boosting iterations vs Depth of Tree: Boosting

| | 10 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| 1 | 82.029178 | 82.824934 | 83.156499 | 83.421751 | 83.222812 | 83.687003 |
| 2 | 83.090186 | 83.355438 | 83.952255 | 84.018568 | 83.952255 | 83.819629 |
| 3 | 84.084881 | 83.355438 | 83.620690 | 83.222812 | 83.156499 | 83.355438 |
| 5 | 83.156499 | 83.222812 | 82.692308 | 82.360743 | 81.432361 | 81.498674 |
| 10 | 81.366048 | 80.769231 | 80.238727 | 79.111406 | 80.371353 | 79.442971 |

Testing Accuracies for Number of boosting iterations vs Depth of Tree: Boosting

| | 10 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| 1 | 81.118881 | 82.470862 | 83.123543 | 83.216783 | 83.543124 | 83.543124 |
| 2 | 81.724942 | 82.703963 | 82.377622 | 83.170163 | 83.263403 | 82.937063 |
| 3 | 81.678322 | 82.097902 | 82.564103 | 83.123543 | 83.170163 | 83.496503 |
| 5 | 82.610723 | 83.030303 | 82.424242 | 82.284382 | 81.818182 | 81.165501 |
| 10 | 80.885781 | 80.000000 | 79.114219 | 78.275058 | 79.067599 | 78.601399 |

Train Accuracies for Depth of Tree vs Number of boosting iterations: Boosting

| | 1 | 2 | 3 | 5 | 10 |
|---|---|---|---|---|---|
| 10 | 81.280 | 82.976 | 83.252 | 84.260 | 87.752 |
| 20 | 82.876 | 83.568 | 83.572 | 84.720 | 91.364 |
| 40 | 83.436 | 83.876 | 84.280 | 85.760 | 95.628 |
| 60 | 83.524 | 84.144 | 84.516 | 86.644 | 96.696 |
| 80 | 83.548 | 84.188 | 84.836 | 87.460 | 96.716 |
| 100 | 83.556 | 84.436 | 85.192 | 88.144 | 96.716 |

Development Accuracies for Depth of Tree vs Number of boosting iterations: Boosting

| | 1 | 2 | 3 | 5 | 10 |
|---|---|---|---|---|---|
| 10 | 82.029178 | 83.090186 | 84.084881 | 83.156499 | 82.029178 |
| 20 | 82.824934 | 83.355438 | 83.355438 | 83.222812 | 81.034483 |
| 40 | 83.156499 | 83.952255 | 83.620690 | 82.824934 | 78.978780 |
| 60 | 83.421751 | 84.018568 | 83.222812 | 82.360743 | 79.575597 |
| 80 | 83.222812 | 83.952255 | 83.156499 | 81.432361 | 80.371353 |
| 100 | 83.687003 | 83.819629 | 83.355438 | 81.697613 | 79.774536 |

Testing Accuracies for Depth of Tree vs Number of boosting iterations: Boosting

| | 1 | 2 | 3 | 5 | 10 |
|---|---|---|---|---|---|
| 10 | 81.118881 | 81.724942 | 81.678322 | 82.610723 | 81.072261 |
| 20 | 82.470862 | 82.703963 | 82.097902 | 83.030303 | 80.606061 |
| 40 | 83.123543 | 82.377622 | 82.564103 | 82.517483 | 78.834499 |
| 60 | 83.216783 | 83.170163 | 83.123543 | 82.237762 | 78.834499 |
| 80 | 83.543124 | 83.263403 | 83.170163 | 81.864802 | 78.228438 |
| 100 | 83.543124 | 82.890443 | 83.449883 | 81.258741 | 79.020979 |

Observations

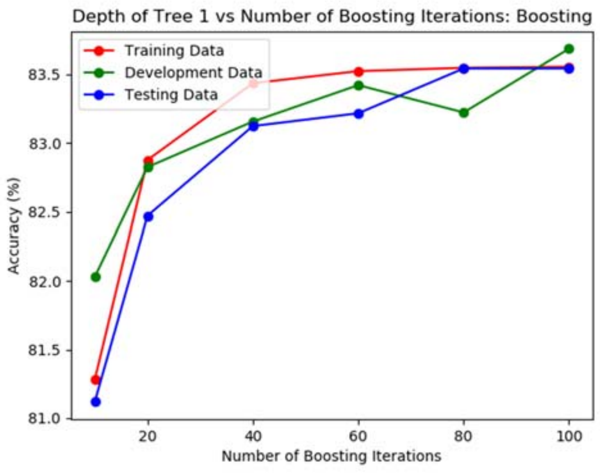
Number of boosting iterations vs Depth of Tree

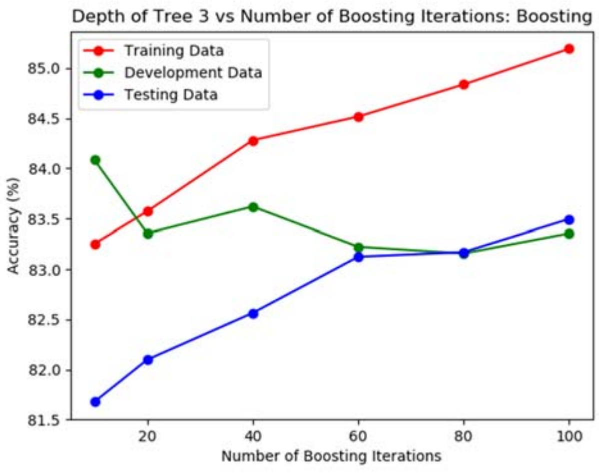
- For training data, when the number of boosting iterations increases the accuracy increases.
- For development data, for depths 1 accuracy increases and for the other depths, the accuracy decreases.
- For testing data, for depths 1 and 3, accuracy increases and for the other depths, the accuracy decreases.

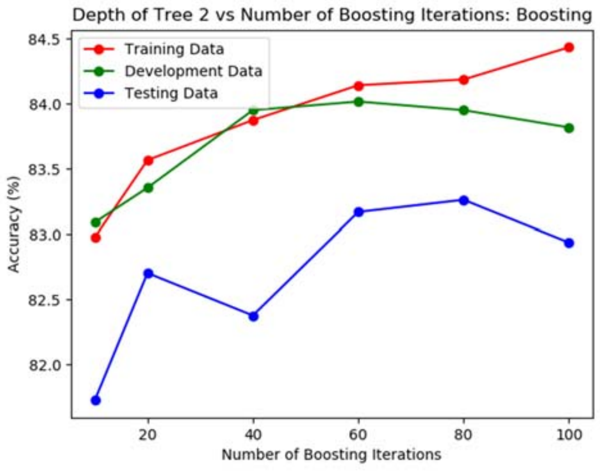
Depth of Tree vs Number of boosting iterations:

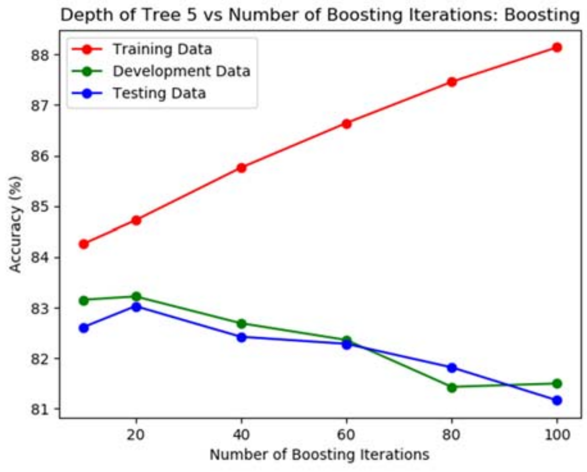
- For training data, when the depth increases the accuracy increases.
- For development data, when the depth increases the accuracy decreases.
- For testing data, when the depth increases the accuracy decreases.

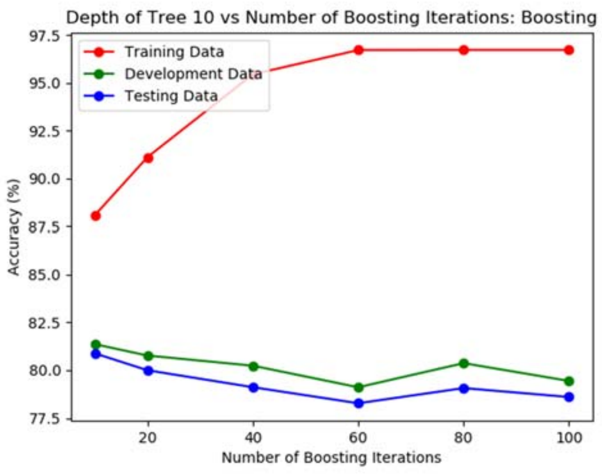
Number of boosting iterations vs Depth of Tree



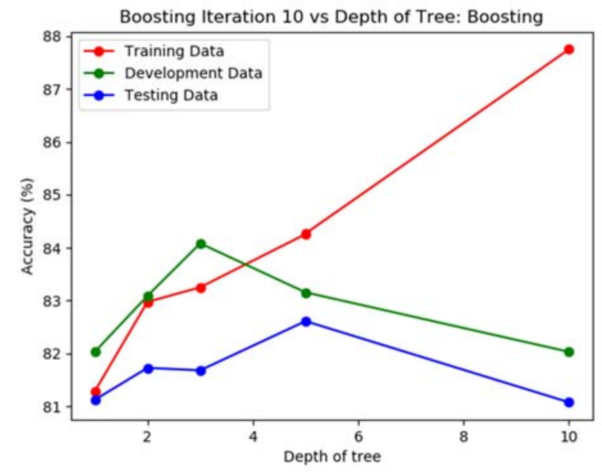


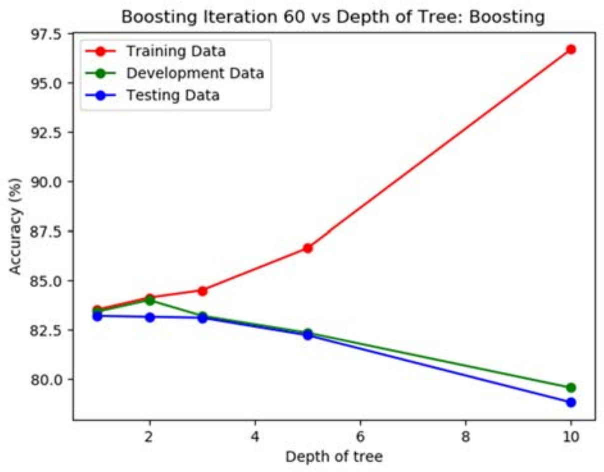


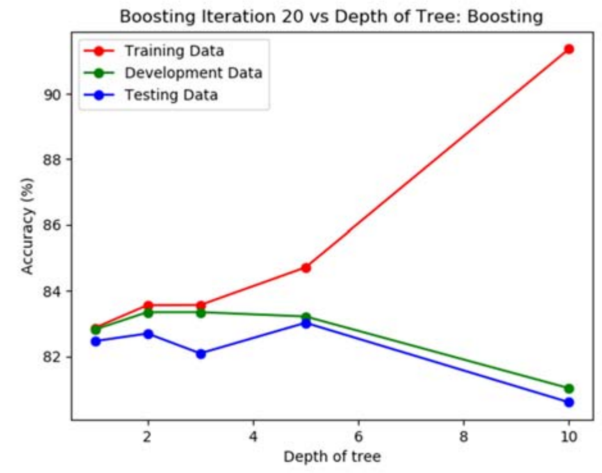


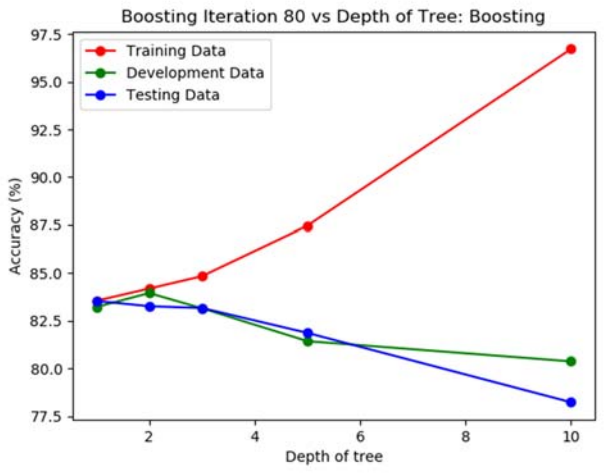


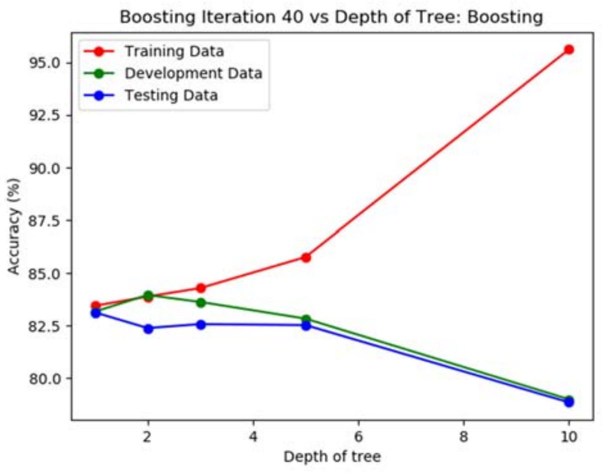
Depth of Tree vs Number of boosting iterations:

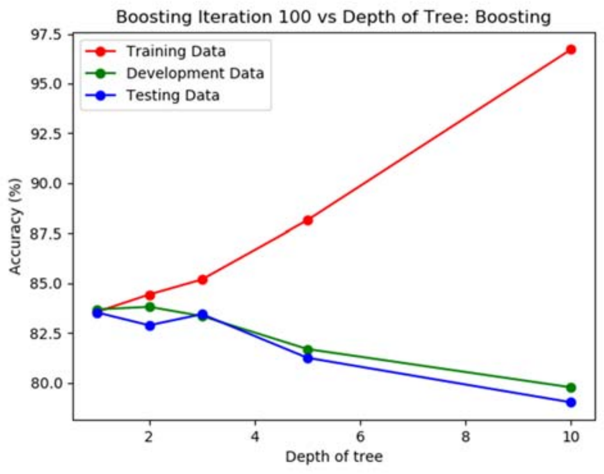






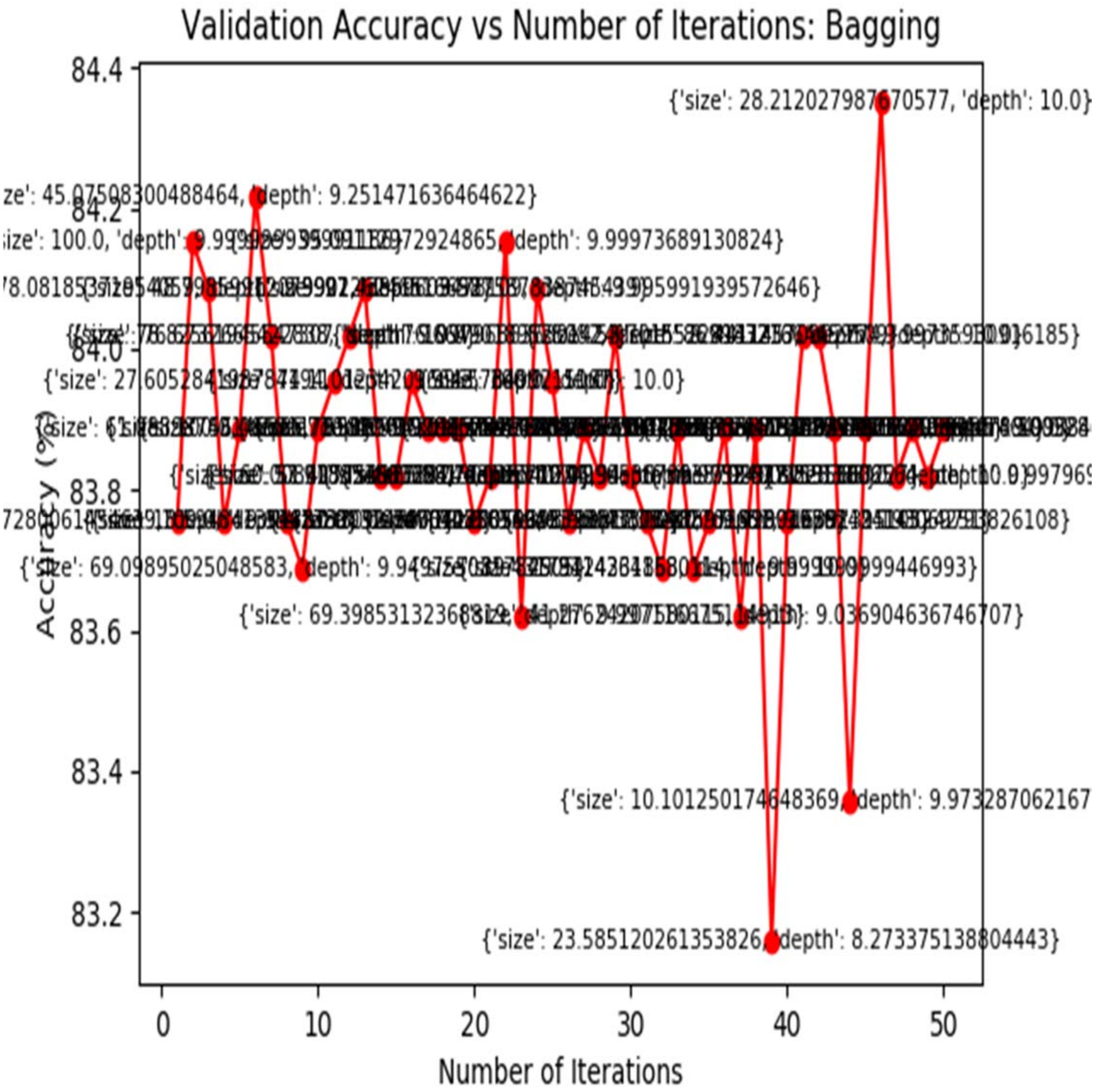




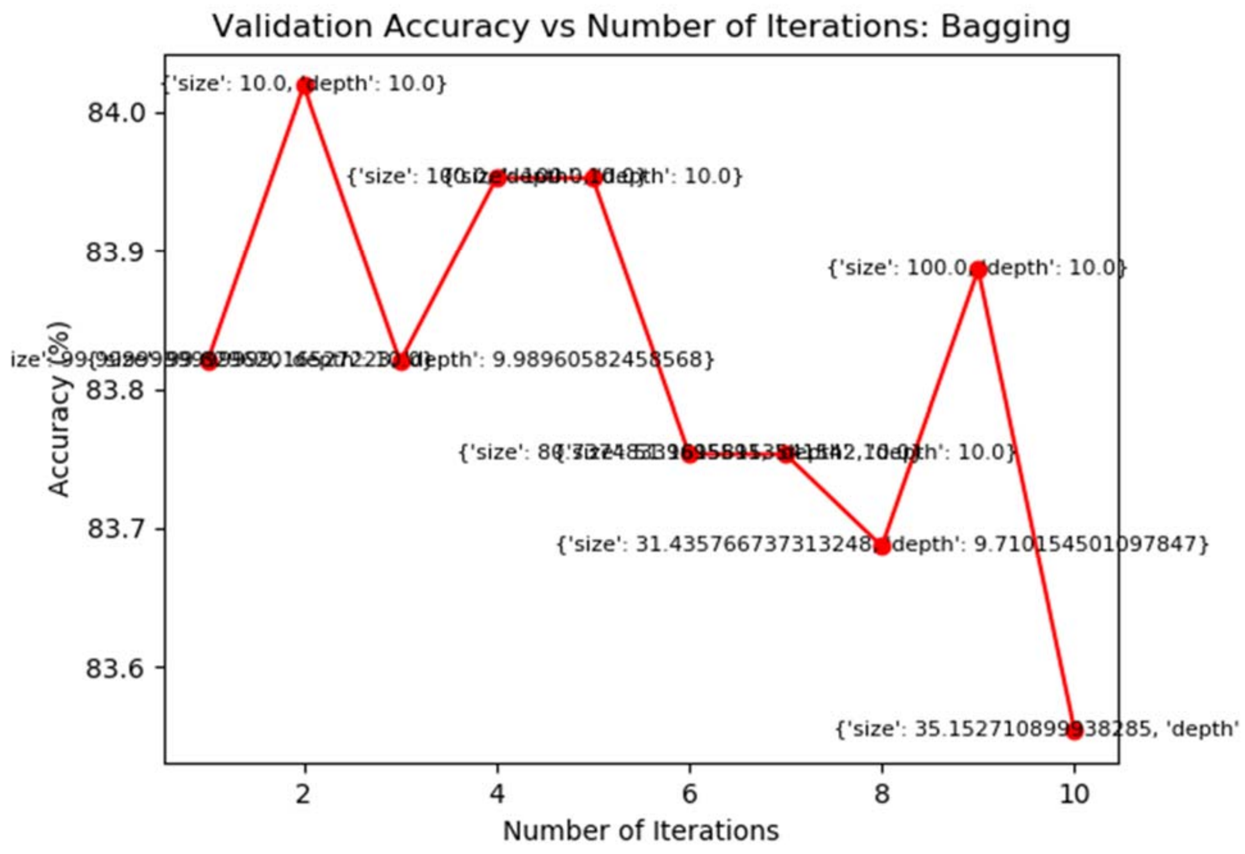


**7. Automatic hyper-parameter tuning via Bayesian Optimization.**

50 iterations for Bagging



10 iterations for Bagging

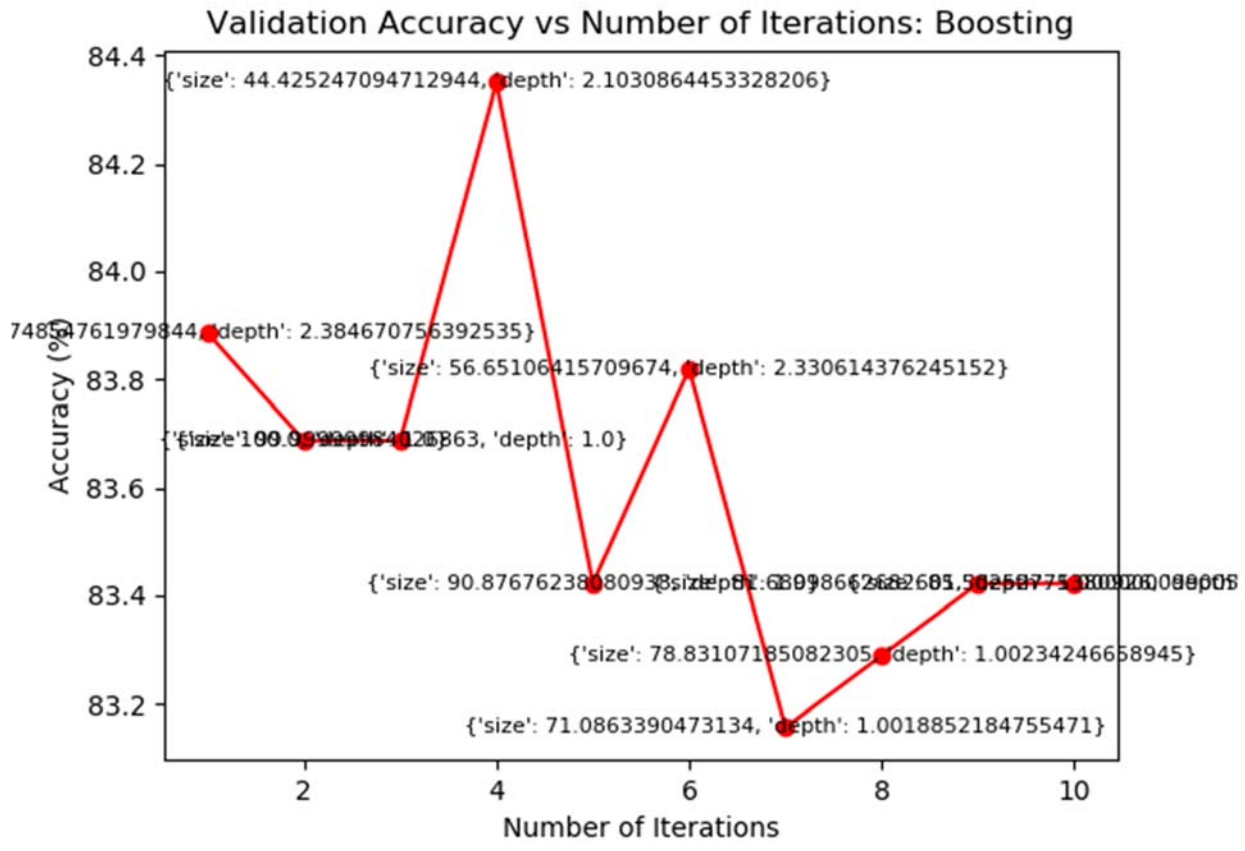


Bagging Bayesian Optimisation:

Accuracies: Bagging

| | Depth of Tree | Ensemble Size | Validation Accuracy |
|---|---|---|---|
| 0 | 10.000000 | 100.000000 | 83.819629 |
| 1 | 10.000000 | 10.000000 | 84.018568 |
| 2 | 9.989606 | 99.829620 | 83.819629 |
| 3 | 10.000000 | 100.000000 | 83.952255 |
| 4 | 10.000000 | 100.000000 | 83.952255 |
| 5 | 10.000000 | 80.737483 | 83.753316 |
| 6 | 10.000000 | 51.161582 | 83.753316 |
| 7 | 9.710155 | 31.435767 | 83.687003 |
| 8 | 10.000000 | 100.000000 | 83.885942 |
| 9 | 10.000000 | 35.152711 | 83.554377 |

10 iterations for Boosting



Boosting Bayesian Optimisation:

Accuracies: Boosting

| | Depth of Tree | Ensemble Size | Validation Accuracy |
|---|---|---|---|
| 0 | 2.384671 | 61.174855 | 83.885942 |
| 1 | 1.000000 | 100.000000 | 83.687003 |
| 2 | 1.000000 | 100.000000 | 83.687003 |
| 3 | 2.103086 | 44.425247 | 84.350133 |
| 4 | 1.000000 | 90.876762 | 83.421751 |
| 5 | 2.330614 | 56.651064 | 83.819629 |
| 6 | 1.001885 | 71.086339 | 83.156499 |
| 7 | 1.002342 | 78.831072 | 83.289125 |
| 8 | 1.000000 | 81.689987 | 83.421751 |
| 9 | 1.000000 | 85.502528 | 83.421751 |