# CS 5350/6350: Machine Learning Fall 2021

Homework 2

Handed out: 28 Sep, 2021
Due date: 11:59pm, 19 Oct, 2021

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free to discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You do not need to include original problem descriptions in your solutions. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 20 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- *Your code should run on the CADE machines.* You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.

  You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.
- Please do not hand in binary files! We will *not* grade binary submissions.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.

## 1 Paper Problems [40 points + 8 bonus]

1. [5 points] We have derived the PAC guarantee for consistent learners (namely, the learners can produce a hypothesis that can 100% accurately classify the training data). The PAC guarantee is described as follows. Let $H$ be the hypothesis space used by our algorithm. Let $C$ be the concept class we want to apply our learning algorithm to search for a target function in $C$. We have shown that, with probability at least $1-\delta$, a hypothesis $h \in H$ that is consistent with a training set of $m$ examples will have the generalization error $\mathrm{err}_D(h) < \epsilon$ if

$$m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big).$$

   (a) [2 points] Suppose we have two learning algorithms $L_1$ and $L_2$, which use hypothesis spaces $H_1$ and $H_2$ respectively. We know that $H_1$ is larger than $H_2$, i.e., $|H_1| > |H_2|$. For each target function in $C$, we assume both algorithms can find a hypothesis consistent with the training data.

i. [1 point] According to Occam's Razor principle, which learning algorithm's result hypothesis do you prefer? Why?

According to Occam's Razor principle we would prefer $L_2$'s result hypothesis since $H_2$ is smaller.

ii. [1 point] How is this principle reflected in our PAC guarantee? Please use the above inequality to explain why we will prefer the corresponding result hypothesis.

Since $H_2$ is smaller we will need less training examples to achieve a generalization error $< \epsilon$ with probability $\geq 1 - \delta$.

(b) [3 points] Let us investigate algorithm $L_1$. Suppose we have $n$ input features, and the size of the hypothesis space used by $L_1$ is $3^n$. Given $n = 10$ features, if we want to guarantee a 95% chance of learning a hypothesis of at least 90% generalization accuracy, how many training examples at least do we need for $L_1$?

$$m > \frac{1}{.1}(\ln 3^{10} + \ln \frac{1}{.05})) = 139.8$$

This means that we will need at least 140 training examples for $L_1$.

2. [5 points] In our lecture about AdaBoost algorithm, we introduced the definition of weighted error in each round $t$,

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\Big(\sum_{i=1}^{m} D_t(i) y_i h_t(x_i)\Big)$$

where $D_t(i)$ is the weight of $i$-th training example, and $h_t(x_i)$ is the prediction of the weak classifier learned round $t$. Note that both $y_i$ and $h_t(x_i)$ belong to $\{1, -1\}$. Prove that equivalently,

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i).$$

Here is our starting formula:

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\Big(\sum_{i=1}^{m} D_t(i) y_i h_t(x_i)\Big)$$

We know that $y_i h_t(x_i) = 1$ when $y_i = h_t(x_i)$ and $y_i h_t(x_i) = -1$ when $y_i \neq h_t(x_i)$

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\Big(\sum_{y_i = h_t(x_i)} D_t(i) - \sum_{y_i \neq h_t(x_i)} D_t(i)\Big)$$

We know that also know all the weights should add to one so $\sum_{i=1}^{m} D_t(i) = 1$

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\Big((1 - \sum_{y_i \neq h_t(x_i)} D_t(i)) - \sum_{y_i \neq h_t(x_i)} D_t(i)\Big)$$

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\Big(1 - 2 \sum_{y_i \neq h_t(x_i)} D_t(i)\Big)$$

$$\epsilon_t = \frac{1}{2} - \frac{1}{2} + \sum_{y_i \neq h_t(x_i)} D_t(i)$$

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i)$$

3. [20 points] Can you figure out an equivalent linear classifier for the following Boolean functions? Please point out what the weight vector, the bias parameter and the hyperplane are. Note that the hyperplane is determined by an equation. If you cannot find out a linear classifier, please explain why, and work out some feature mapping such that, after mapping all the inputs of these functions into a higher dimensional space, there is a hyperplane that well separates the inputs; please write down the separating hyperplane in the new feature space.

   (a) [2 point] $f(x_1, x_2, x_3) = x_1 \land \neg x_2 \land \neg x_3$

   $$x_1 - x_2 - x_3 \geq 1$$

   $\mathbf{w} = [1, -1, -1], b = -1$

   (b) [2 point] $f(x_1, x_2, x_3) = \neg x_1 \lor \neg x_2 \lor \neg x_3$

   $$-x_1 - x_2 - x_3 \geq -2$$

   $\mathbf{w} = [-1, -1, -1], b = 2$

   (c) [8 points] $f(x_1, x_2, x_3, x_4) = (x_1 \lor x_2) \land (x_3 \lor x_4)$
   We can map these boolean values into another space. We define $y_1 = x_1 \lor x_2$ and $y_2 = x_3 \lor x_4$.

   (d) [8 points] $f(x_1, x_2) = (x_1 \land x_2) \lor (\neg x_1 \land \neg x_2)$

4. [**Bonus**] [8 points] Given two vectors $\mathbf{x} = [x_1, x_2]$ and $\mathbf{y} = [y_1, y_2]$, find a feature mapping $\phi(\cdot)$ for each of the following functions, such that the function is equal to the inner product between the mapped feature vectors, $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$. For example, $(\mathbf{x}^\top \mathbf{y})^0 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1]$ and $\phi(\mathbf{y}) = [1]$; $(\mathbf{x}^\top \mathbf{y})^1 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = \mathbf{x}$ and $\phi(\mathbf{y}) = \mathbf{y}$.

   (a) [2 points] $(\mathbf{x}^\top \mathbf{y})^2$

   (b) [2 points] $(\mathbf{x}^\top \mathbf{y})^3$

   (c) [4 points] $(\mathbf{x}^\top \mathbf{y})^k$ where $k$ is any positive integer.

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| 1 | -1 | 2 | 1 |
| 1 | 1 | 3 | 4 |
| -1 | 1 | 0 | -1 |
| 1 | 2 | -4 | -2 |
| 3 | -1 | -1 | 0 |

Table 1: Linear regression training data.

5. [10 points] Suppose we have the training data shown in Table 1, from which we want to learn a linear regression model, parameterized by a weight vector $\mathbf{w}$ and a bias parameter $b$.

   (a) [1 point] Write down the LMS (least mean square) cost function $J(\mathbf{w}, b)$.

   $$J(\mathbf{w}, b) = \frac{1}{2}\sum_{i=1}^{m}(y_i - (\mathbf{w}^T\mathbf{x}_i + b))^2$$

   (b) [3 points] Calculate the gradient $\frac{\nabla J}{\nabla \mathbf{w}}$ and $\frac{\nabla J}{\nabla b}$ when $\mathbf{w} = [-1, 1, -1]^\top$ and $b = -1$. We know that $\frac{\delta J}{\delta w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$ and $\frac{\delta J}{\delta b} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)$

   $$\frac{\nabla J}{\nabla \mathbf{w}} = [-22, 16, -56]$$

   $$\frac{\nabla J}{\nabla b} = -10$$

   (c) [3 points] What are the optimal $\mathbf{w}$ and $\mathbf{b}$ that minimize the cost function? Let's start by defining the augmented weight vector $\mathbf{w}' = [\mathbf{w}, b]^T$. From here we can redefine the the gradient as:

   $$\frac{\nabla J}{\nabla w'} = -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}')$$

   where $\mathbf{X}$ is a $n$ x $d$ matrix where the rows represent an example and a column a feature and $y$ is a $n$-dimensional vector representing the output for each example. Now we can set the gradient to $\mathbf{0}$ and solve for the $\mathbf{w}'$.

   $$\mathbf{0} = -\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w}' \Rightarrow \mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}' \Rightarrow (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = [1, 1, 1 - 1] = \mathbf{w}'$$

   $$\mathbf{w} = [1, 1, 1], b = -1$$

   (d) [3 points] Now, we want to use stochastic gradient descent to minimize $J(\mathbf{w}, b)$. We initialize $\mathbf{w} = \mathbf{0}$ and $b = 0$. We set the learning rate $r = 0.1$ and sequentially go through the 5 training examples. Please list the stochastic gradient in each step and the updated $\mathbf{w}$ and $b$.
   We again use the definition of $\mathbf{w}'$ from our previous problem. For each step $i$ we

calculate $\nabla J = -X_i(y_i - w' \cdot X_i), w'_{t+1} = w'_t - r * \nabla J$ (I used numpy to do all my calculations).

$$\nabla J = [-1, 1, -2, -1], \mathbf{w}'_1 = [0.1, -0.1, 0.2, 0.1]$$

$$\nabla J = [-3.3, -3.3, -9.9, -3.3], \mathbf{w}'_2 = [0.43, 0.23, 1.19, 0.43]$$

$$\nabla J = [-1.23, 1.23, 0., 1.23], \mathbf{w}'_3 = [0.553, 0.107, 1.19, 0.307]$$

$$\nabla J = [-1.686, -3.372, 6.744, -1.686], \mathbf{w}'_4 = [0.7216, 0.4442, 0.5156, 0.4756]$$

$$\nabla J = [5.042, -1.681, -1.681, 1.681], \mathbf{w}'_5 = [0.217, 0.612, 0.684, 0.308]$$

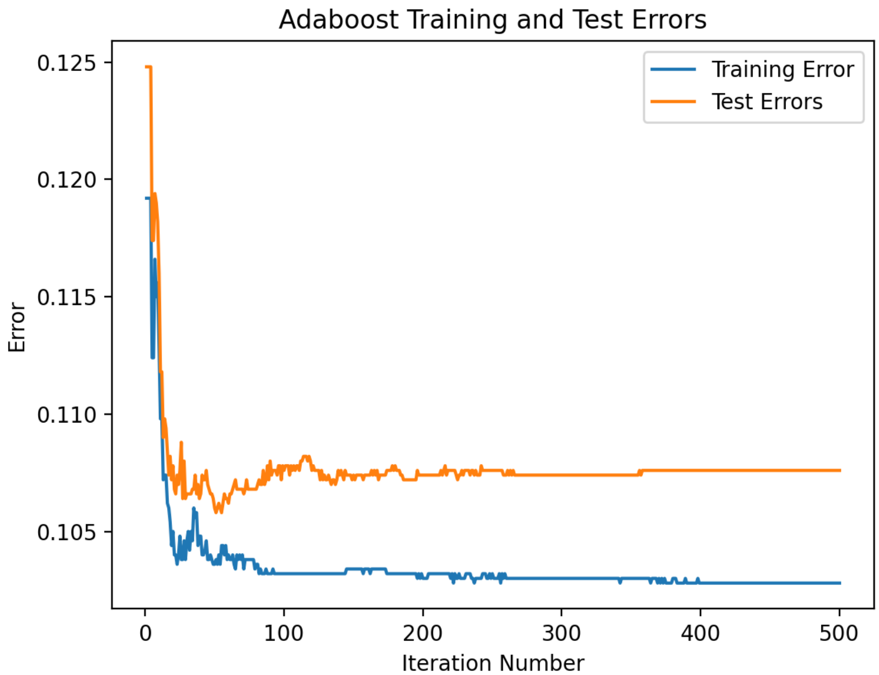
The last round of values are rounded.

# 2 Practice [60 points + 10 bonus]

1. [2 Points] Update your machine learning library. Please check in your implementation of decision trees in HW1 to your GitHub repository. Remember last time you created a folder "Decision Tree". You can commit your code into that folder. Please also supplement README.md with concise descriptions about how to use your code to learn decision trees (how to call the command, set the parameters, etc). Please create two folders "Ensemble Learning" and "Linear Regression" in the same level as the folder "Decision Tree".

2. [36 points] We will implement the boosting and bagging algorithms based on decision trees. Let us test them on the bank marketing dataset in HW1 (bank.zip in Canvas). We use the same approach to convert the numerical features into binary ones. That is, we choose the media (NOT the average) of the attribute values (in the training set) as the threshold, and examine if the feature is bigger (or less) than the threshold. For simplicity, we treat "unknown" as a particular attribute value, and hence we do not have any missing attributes for both training and test.

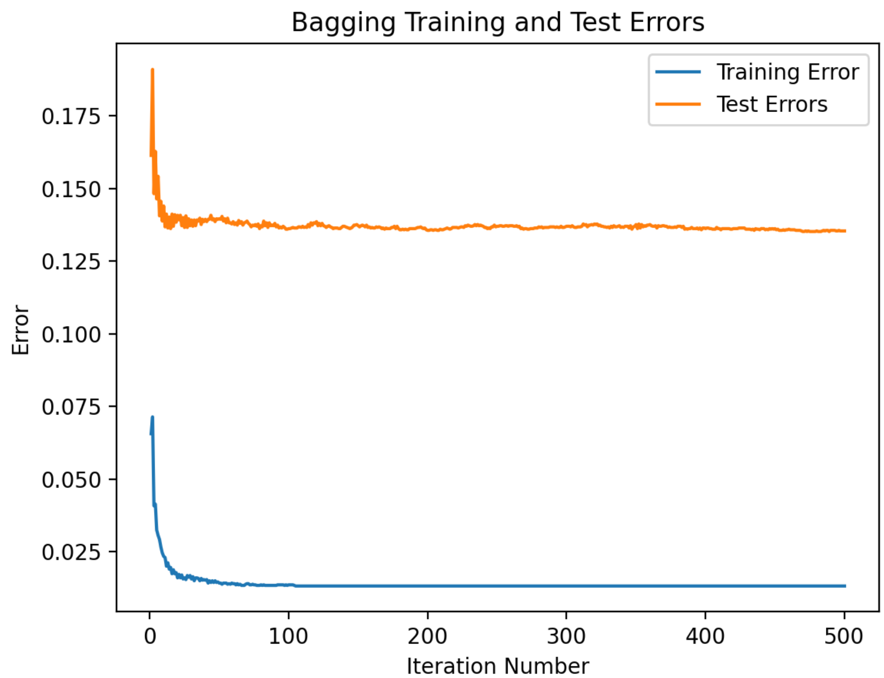
   (a) [8 points] Modify your decision tree learning algorithm to learn decision stumps — trees with only two levels. Specifically, compute the information gain to select the best feature to split the data. Then for each subset, create a leaf node. Note that your decision stumps must support weighted training examples. Based on your decision stump learning algorithm, implement AdaBoost algorithm. Vary the number of iterations T from 1 to 500, and examine the training and test errors. You should report the results in two figures. The first figure shows how the training and test errors vary along with T. The second figure shows the training and test errors of all the decision stumps learned in each iteration. What can you observe and conclude? You have had the results for a fully expanded decision tree in HW1. Comparing them with Adaboost, what can you observe and conclude?





Adaboost seems to have a better testing error but not a better training error so it seems to generalize better.

(b) [8 points] Based on your code of the decision tree learning algorithm (with information gain), implement a Bagged trees learning algorithm. Note that each tree should be fully expanded — no early stopping or post pruning. Vary the number of trees from 1 to 500, report how the training and test errors vary along with the tree number in a figure. Overall, are bagged trees better than a single tree? Are bagged trees better than Adaboost?



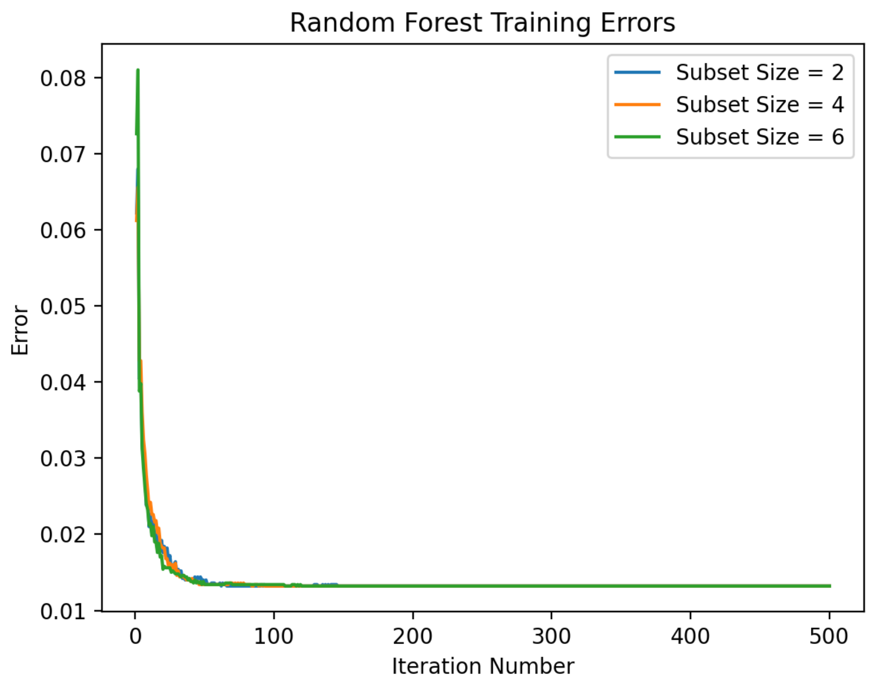

The bagging algorithm seems pretty similar to a fully expanded tree (maybe a little bit better in its testing error). However it doesn't generalize as well as Adaboost.

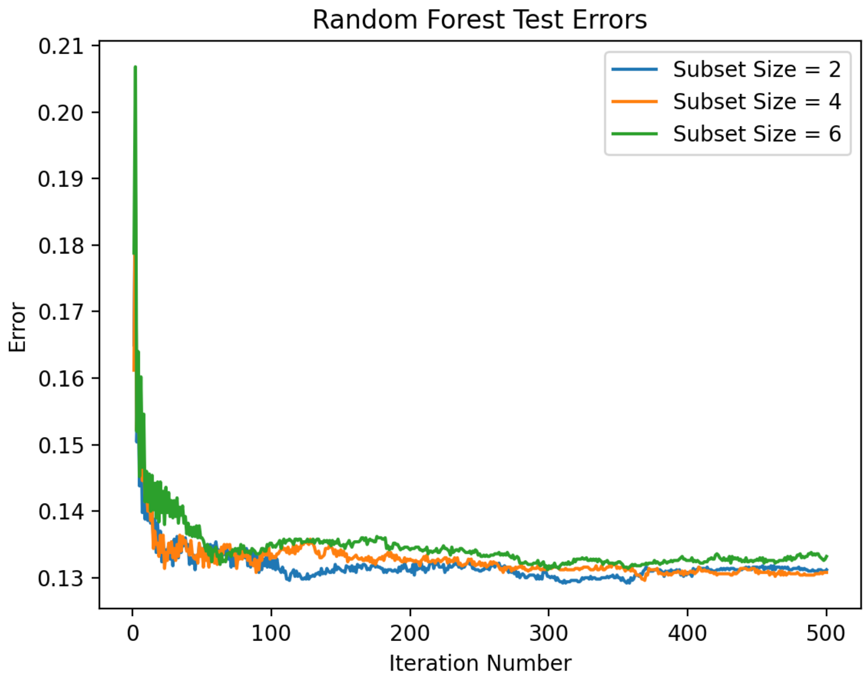
(c) [6 points] Through the bias and variance decomposition, we have justified why the bagging approach is more effective than a single classifier/predictor. Let us verify it in real data. Experiment with the following procedure.

- REPEAT for 100 times
- [STEP 1] Sample $1,000$ examples *uniformly without replacement* from the training datset
- [STEP 2] Run your bagged trees learning algorithm based on the $1,000$ training examples and learn 500 trees.
- END REPEAT
- Now you have 100 bagged predictors in hand. For comparison, pick the first tree in each run to get 100 fully expanded trees (i.e. single trees).
- For each of the test example, compute the predictions of the 100 single trees. Take the average, subtract the ground-truth label, and take square to compute the bias term (see the lecture slides). Use all the predictions to compute the sample variance as the approximation to the variance term (if you forget what the sample variance is, check it out here). You now obtain the bias and variance terms of a single tree learner for one test example. You will need to compute them for all the test examples and then take average as your final estimate of the bias and variance terms for the single decision tree

learner. You can add the two terms to obtain the estimate of the general squared error (that is, expected error w.r.t test examples). Now use your 100 bagged predictors to do the same thing and estimate the general bias and variance terms, as well as the general squared error. Comparing the results of the single tree learner and the bagged trees, what can you conclude? What causes the difference?

(d) [8 points] Implement the random forest algorithm as we discussed in our lecture. Vary the number of random trees from 1 to 500. Note that you need to modify your tree learning algorithm to randomly select a subset of features before each split. Then use the information gain to select the best feature to split. Vary the size of the feature subset from $\{2, 4, 6\}$. Report in a figure how the training and test errors vary along with the number of random trees for each feature subset size setting. How does the performance compare with bagged trees?





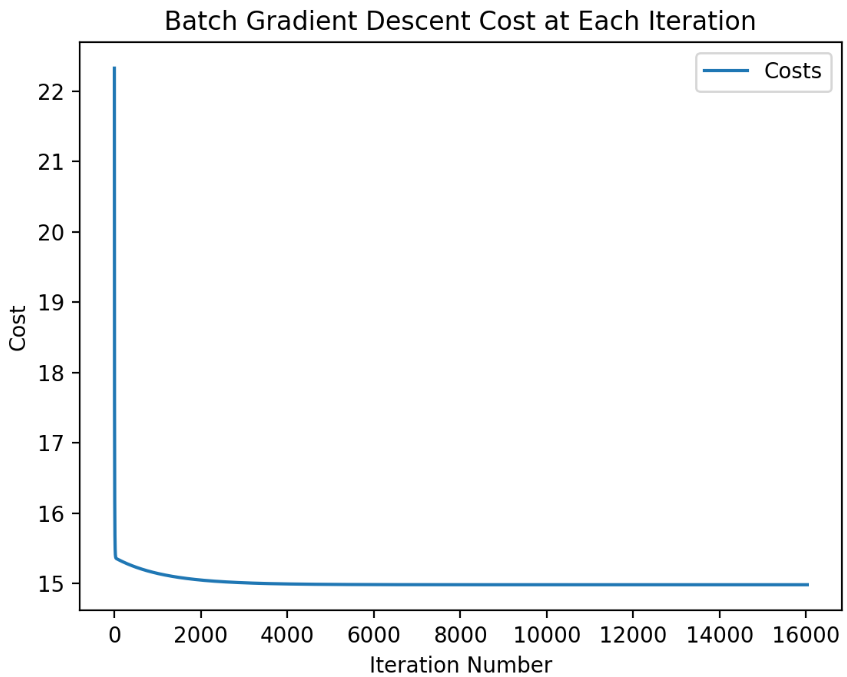

Random Forest seems to do a little better than the Bagging Algorithm

(e) [6 points] Following (c), estimate the bias and variance terms, and the squared error for a single random tree and the whole forest. Comparing with the bagged trees, what do you observe? What can you conclude?

3. [**Bonus**][10 points] In practice, to confirm the performance of your algorithm, you need to find multiple datasets for test (rather than one). You need to extract and process data by yourself. Now please use the credit default dataset in UCI repository https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Randomly choose 24000 examples for training and the remaining 6000 for test. Feel free to deal with continuous features. Run bagged trees, random forest, and Adaboost with decision stumps algorithms for 500 iterations. Report in a figure how the training and test errors vary along with the number of iterations, as compared with a fully expanded single decision tree. Are the results consistent with the results you obtained from the bank dataset?

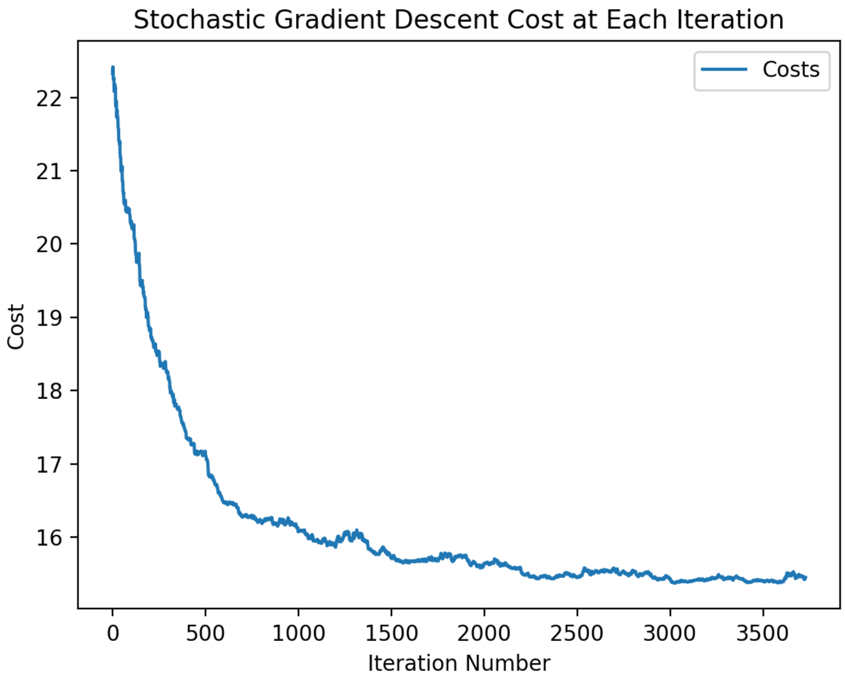
4. [22 points] We will implement the LMS method for a linear regression task. The dataset is from UCI repository (`https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test`). The task is to predict the real-valued SLUMP of the concrete, with 7 features. The features and output are listed in the file "concrete/data-desc.txt". The training data are stored in the file "concrete/train.csv", consisting of 53 examples. The test data are stored in "concrete/test.csv", and comprise of 50 examples. In both the training and testing datasets, feature values and outputs are separated by commas.

   (a) [8 points] Implement the batch gradient descent algorithm, and tune the learning rate $r$ to ensure the algorithm converges. To examine convergence, you can watch the norm of the weight vector difference, $\|w_t - w_{t-1}\|$, at each step $t$. if $\|w_t - w_{t-1}\|$ is less than a tolerance level, say, $10^{-6}$, you can conclude that it converges. You can initialize your weight vector to be $\mathbf{0}$. Please find an appropriate $r$ such that the algorithm converges. To tune $r$, you can start with a relatively big value, say, $r = 1$, and then gradually decrease $r$, say $r = 0.5, 0.25, 0.125, \ldots$, until you see the convergence. Report the learned weight vector, and the learning rate $r$. Meanwhile, please record the cost function value of the training data at each step, and then draw a figure shows how the cost function changes along with steps. Use your final weight vector to calculate the cost function value of the test data.




Final Weight Vector(bgd):
[ 0.89969405 0.78539622 0.85007385 1.29819895 0.12974519 1.57107959 0.99780524 -0.01521448]
Final Test Cost(bgd): 23.360587522223284

(b) [8 points] Implement the stochastic gradient descent (SGD) algorithm. You can initialize your weight vector to be $\mathbf{0}$. Each step, you randomly sample a training example, and then calculate the stochastic gradient to update the weight vector. Tune the learning rate $r$ to ensure your SGD converges. To check convergence, you can calculate the cost function of the training data after each stochastic gradient update, and draw a figure showing how the cost function values vary along with the number of updates. At the beginning, your curve will oscillate a lot. However, with an appropriate $r$, as more and more updates are finished, you will see the cost function tends to converge. Please report the learned weight vector, and the learning rate you chose, and the cost function value of the test data with your learned weight vector.




Final Weight Vector(sgd):
[ 0.1330348 -0.0961236 -0.0408294 0.58685032 -0.0493948 0.42395254 0.16574373 0.00766225]
Final Test Cost(sgd): 21.621959275431184

(c) [6 points] We have discussed how to calculate the optimal weight vector with an analytical form. Please calculate the optimal weight vector in this way. Comparing with the weight vectors learned by batch gradient descent and stochastic gradient descent, what can you conclude? Why?
Analytical $\mathbf{w}$ = [ 0.90056451 0.78629331 0.85104314 1.29889413 0.12989067 1.57224887 0.99869359 -0.01519667]
Batch Gradient Descent found the closer weight vector.