CS 5350/6350: Machine Learning Spring 2020

Homework 2

Handed out: 11 Feb, 2019
Due date: 11:59pm, 25 Feb, 2019

1 Paper Problems [40 points + 8 bonus]

1. (a) i. L_2. Because it has less elements and so needs a small number of training examples to give a hypothesis with desired accuracy.

   ii. In the inequality $m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big)$ we see that if $|H|$ increases, then m should be increased. That is, with a same accuracy parameter δ and same error ϵ we have
$$m > \frac{1}{\epsilon}\big(\log(|H_1|) + \log\frac{1}{\delta}\big) \geq \frac{1}{\epsilon}\big(\log(|H_2|) + \log\frac{1}{\delta}\big).$$
   This suggests the smaller $|H|$, the smaller m is required which is exactly the Occam Razor principal.

   (b) Here we have $|H| = 3^{10}$, $\delta = 0.05$ (as we have $1-\delta = 0.95$) and $\epsilon = 0.1$ (as generalization accuracy is 0.9). So, according to the inequality $m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big)$ we get $m > \frac{1}{0.1}\big(\log(3^{10}) + \log\frac{1}{0.05}\big)$ or equivalently $m > 139.82$, i.e., m should be at least 140.

2. As $\{D_t(i) : i = 1, \ldots, m\}$ is the sequence of weights, we know that $\sum_{i=1}^m D_t(i) = 1$ for each t. Thus noting the fact that $y_i h_t(x_i) = 1$ when $y_i = h_t(x_i)$ and $y_i h_t(x_i) = -1$ when $y_i \neq h_t(x_i)$ we can write $1 = \sum_{i=1}^m D_t(i) = \sum_{y_i = h_t(x_i)} D_t(i) + \sum_{y_i \neq h_t(x_i)} D_t(i)$, and so
$$\sum_{y_i = h_t(x_i)} D_t(i) = 1 - \sum_{y_i \neq h_t(x_i)} D_t(i) \tag{1}$$
   Now if $\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{i=1}^m D_t(i) y_i h_t(x_i)\big)$, then $1 - 2\epsilon_t = \sum_{i=1}^m D_t(i) y_i h_t(x_i)$, and applying

(1) we get

$$\begin{aligned}
1-2\epsilon_t &= \sum_{i=1}^m D_t(i)y_ih_t(x_i) = \sum_{y_i=h_t(x_i)} D_t(i)y_ih_t(x_i) - \sum_{y_i\neq h_t(x_i)} D_t(i)y_ih_t(x_i)\\
&= \sum_{y_i=h_t(x_i)} D_t(i) - \sum_{y_i\neq h_t(x_i)} D_t(i) = 1 - \sum_{y_i\neq h_t(x_i)} D_t(i) - \sum_{y_i\neq h_t(x_i)} D_t(i)\\
&= 1-2\sum_{y_i\neq h_t(x_i)} D_t(i),
\end{aligned}$$

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

Conversely, let $\epsilon_t = \sum_{y_i\neq h_t(x_i)} D_t(i)$. Then again using (1) we have

$$\begin{aligned}
1-2\epsilon_t &= 1-2\sum_{y_i\neq h_t(x_i)} D_t(i) = \big(1-\sum_{y_i\neq h_t(x_i)} D_t(i)\big) - \sum_{y_i\neq h_t(x_i)} D_t(i)\\
&= \sum_{y_i=h_t(x_i)} D_t(i) - \sum_{y_i\neq h_t(x_i)} D_t(i) = \sum_{y_i=h_t(x_i)} D_t(i)y_ih_t(x_i) - \sum_{y_i\neq h_t(x_i)} D_t(i)y_ih_t(x_i)\\
&= \sum_{i=1}^m D_t(i)y_ih_t(x_i),
\end{aligned}$$

and so $\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{i=1}^m D_t(i)y_ih_t(x_i)\big)$.

3. (a) $f(x_1,x_2,x_3) = x_1 \wedge \neg x_2 \wedge \neg x_3$

   $x_1 + (1-x_2) + (1-x_3) \geq 3$ which implies $x_1 - x_2 - x_3 \geq 1$. So, $x_1 - x_2 - x_3 = 1$ is a linear classifier.

   (b) $f(x_1,x_2,x_3) = \neg x_1 \vee \neg x_2 \vee \neg x_3$

   Since $\neg x_1 \vee \neg x_2 \vee \neg x_3 = \neg(x_1 \wedge x_2 \wedge x_3)$, we can use $\neg(x_1+x_2+x_3 \geq 3)$ which is $x_1+x_2+x_3 < 3$ or equivalently $x_1+x_2+x_3 \leq 2$. So, $x_1+x_2+x_3 = 2$ is a linear classifier.

   (c) $f(x_1,x_2,x_3,x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$

   We need to have $x_1 + x_2 \geq 1$ or $x_3 + x_4 \geq 1$. This does not give a linear classifier. So we can use a feature mapping like $T : \mathbb{R}^4 \to \mathbb{R}^5$ defined by

$$T(x_1,x_2,x_3,x_4) = (x_1,x_2,x_3,x_4,(x_1+x_2)(x_3+x_4)),$$

   to map points in such a way that there is a linear classifier. In this case, after mapping by T, the hyperplane $x_5 = \frac{1}{2}$ would be a linear classifier. Note that T is also one-to-one.

(d) $f(x_1, x_2) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

We need to have $x_1 + x_2 \geq 2$ or $x_1 + x_2 \leq 0$. This does not give a linear classifier. So we can use a feature mapping like $T : \mathbb{R}^2 \to \mathbb{R}^2$ defined by

$$T(x_1, x_2) = \Big(\frac{x_1}{1 + (\sqrt[4]{|x_1|} + \sqrt[4]{|x_2|})^4}, \frac{x_2}{1 + (\sqrt[4]{|x_1|} + \sqrt[4]{|x_2|})^4}\Big),$$

to map points in such a way that there is a linear classifier. In this case, after mapping by T, $x_1 + x_2 = \frac{1}{4}$ would be a linear classifier.

Or we can utilise the mapping $\tilde{T} : \mathbb{R}^2 \to \mathbb{R}^4$ induced by T, that is,

$$\tilde{T}(x_1, x_2) = \Big(x_1, x_2, \frac{x_1}{1 + (\sqrt[4]{|x_1|} + \sqrt[4]{|x_2|})^4}, \frac{x_2}{1 + (\sqrt[4]{|x_1|} + \sqrt[4]{|x_2|})^4}\Big),$$

to map points to a 4-dimensional space and find a hyperplane as a linear classifier.

Another linear separator can be, for instance, $S : \mathbb{R}^2 \to \mathbb{R}$ being ruled by

$$S(x_1, x_2) = (x_1 - \frac{1}{2})(x_2 - \frac{1}{2}),$$

which in this case, the point 0 would be a linear classifier. Or we can use a one-to-one embedding $\tilde{S} : \mathbb{R}^2 \to \mathbb{R}^3$ by

$$\tilde{S}(x_1, x_2) = (x_1, x_2, (x_1 - \frac{1}{2})(x_2 - \frac{1}{2})).$$

In this situation, the plane $x_1 + x_2 = 0$ would be a linear classifier.

4. [**Bonus**]

(a) $(\mathbf{x}^\top \mathbf{y})^2 = x_1^2 y_1^2 + 2x_1 x_2 y_2 + y_1^2 y_2^2$. Thus $\phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1x_2, x_2^2]$ and $\phi(\mathbf{y}) = [y_1^2, \sqrt{2}y_1y_2, y_2^2]$ satisfy $(\mathbf{x}^\top \mathbf{y})^2 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$.

(b) $(\mathbf{x}^\top \mathbf{y})^3 = x_1^3 y_1^3 + 3x_1^2 y_1^2 x_2 y_2 + 3x_1 y_1 x_2^2 y_2^2 + x_2^3 y_2^3$. Hence $\phi(\mathbf{x}) = [x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3]$ and $\phi(\mathbf{y}) = [y_1^3, \sqrt{3}y_1^2 y_2, \sqrt{3}y_1 y_2^2, y_2^3]$ satisfy $(\mathbf{x}^\top \mathbf{y})^3 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$.

(c) $(\mathbf{x}^\top \mathbf{y})^k = \sum_{i=0}^{k} \binom{k}{i} x_1^{k-i} y_1^{k-i} x_2^i y_2^i$. Therefore,

$$\phi(\mathbf{x}) = \Big[x_1^k, \sqrt{\binom{k}{1}} x_1^{k-1} x_2, \sqrt{\binom{k}{2}} x_1^{k-2} x_2^2, \sqrt{\binom{k}{3}} x_1^{k-3} x_2^3, \dots, \sqrt{\binom{k}{k-1}} x_1 x_2^{k-1}, x_2^2\Big],$$

and same formula for $\phi(\mathbf{y})$ satisfy $(\mathbf{x}^\top \mathbf{y})^3 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$.

5. (a)

$$\begin{aligned} J(\mathbf{w}, b) &= \tfrac{1}{2}\big(1 - (b + w_1 - w_2 + 2w_3)\big)^2 + \tfrac{1}{2}\big(4 - (b + w_1 + w_2 + 3w_3)\big)^2 \\ &+ \tfrac{1}{2}\big(-1 - (b - w_1 + w_2)\big)^2 + \tfrac{1}{2}\big(-2 - (b + w_1 + 2w_2 - 4w_3)\big)^2 \\ &+ \tfrac{1}{2}\big(0 - (b + 3w_1 - w_2 - w_3)\big)^2. \end{aligned}$$

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.

(b) Since $\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$ we compute partial derivatives as follows:

$$\begin{aligned}\frac{\partial J}{\partial w_1} &= -\big(1-(b+w_1-w_2+2w_3)\big)-\big(4-(b+w_1+w_2+3w_3)\big)\\ &+\big(-1-(b-w_1+w_2)\big)-\big(-2-(b+w_1+2w_2-4w_3)\big)\\ &-3\big(0-(b+3w_1-w_2-w_3)\big) = -4+5b+13w_1-2w_2-2w_3\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial w_2} &= \big(1-(b+w_1-w_2+2w_3)\big)-\big(4-(b+w_1+w_2+3w_3)\big)\\ &-\big(-1-(b-w_1+w_2)\big)-2\big(-2-(b+w_1+2w_2-4w_3)\big)\\ &+\big(0-(b+3w_1-w_2-w_3)\big) = 2+2b-2w_1+8w_2-6w_3\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial w_3} &= -2\big(1-(b+w_1-w_2+2w_3)\big)-3\big(4-(b+w_1+w_2+3w_3)\big)\\ &+4\big(-2-(b+w_1+2w_2-4w_3)\big)+\big(0-(b+3w_1-w_2-w_3)\big)\\ &= -22-2w_1-6w_2+30w_3\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial b} &= -\big(1-(b+w_1-w_2+2w_3)\big)-\big(4-(b+w_1+w_2+3w_3)\big)\\ &-\big(-1-(b-w_1+w_2)\big)-\big(-2-(b+w_1+2w_2-4w_3)\big)\\ &-\big(0-(b+3w_1-w_2-w_3)\big) = -2+5b+5w_1+2w_2\end{aligned}$$

Therefore, $\frac{\nabla J}{\nabla \mathbf{w}} = (\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \frac{\partial J}{\partial w_3}) = (-22, 16, -56)$ and $\frac{\partial J}{\partial b} = -10$.

(c) Setting $\frac{\nabla J}{\nabla \mathbf{w}} = (0,0,0)$ and $\frac{\partial J}{\partial b} = 0$ we get $w_1 = 0.4894$, $w_2 = 0.0957$, $w_3 = 0.766$, $b = -0.1277$.

(d) We employ the formula $w_j^1 = w_j^0 + r(y_j - \mathbf{w}^t\mathbf{x}_i)x_{ij}$ for $1 \le j \le 3$ and $1 \le i \le 5$, $r = 0.1$ and $t = 0,1,2,3,4,5$. We know $\mathbf{w} = 0$ and $b = 0$. So,

$w_1^1 = 0 + 0.1(1 - b^0 - w_1^0 + w_2^0 - 2w_3^0)1 = 0.1 - 0.1w_1^0 - 0.1w_2^0 + 0.2w_3^0 = 0.1$

$w_2^1 = 0 + 0.1(1 - b^0 - w_1^0 + w_2^0 - 2w_3^0)(-1) = 0.1 + 0.1w_1^0 + 0.1w_2^0 - 0.2w_3^0 = -0.1$

$w_3^1 = 0 + 0.1(1 - b^0 - w_1^0 + w_2^0 - 2w_3^0)2 = 0.2 - 0.2w_1^0 - 0.2w_2^0 + 0.4w_3^0 = 0.2$

$b^1 = 0 + 0.1(1 - b^0 - w_1^0 + w_2^0 - 2w_3^0)1 = 0.1$

So the first update is $\mathbf{w}^1 = (0.1, -0.1, 0.2)$ and $b^1 = 0.1$.

$w_1^2 = w_1^1 + 0.1(4 - b^1 - w_1^1 - w_2^1 - 3w_3^1)1 = 0.1 + 0.1(4 - 0.1 - 0.1 + 0.1 - 0.6) = 0.43$

$w_2^2 = w_2^1 + 0.1(4 - b^1 - w_1^1 - w_2^1 - 3w_3^1)1 = -0.1 + 0.1(4 - 0.1 - 0.1 + 0.1 - 0.6) = 0.23$

$w_3^2 = w_3^1 + 0.1(4 - b^1 - w_1^1 - w_2^1 - 3w_3^1)3 = 0.2 + 0.3(4 - 0.1 - 0.1 + 0.1 - 0.6) = 1.19$

$b^2 = b^1 + 0.1(4 - b^1 - w_1^1 - w_2^1 - 3w_3^1)1 = 0.1 + 0.1(4 - 0.1 - 0.1 + 0.1 - 0.6) = 0.43$

So the second update is $\mathbf{w}^2 = (0.43, 0.23, 1.19)$ and $b^2 = 0.43$.

$w_1^3 = w_1^2 + 0.1(-1 - b^2 + w_1^2 - w_2^2)(-1) = 0.43 - 0.1(-1 - 0.43 + 0.43 - 0.23) = 0.553$
$w_2^3 = w_2^2 + 0.1(-1 - b^2 + w_1^2 - w_2^2)1 = 0.23 + 0.1(-1 - 0.43 + 0.43 - 0.23) = 0.107$
$w_3^3 = w_3^2 + 0.1(-1 - b^2 + w_1^2 - w_2^2)0 = 1.19 + 0 = 1.19$
$b^3 = b^2 + 0.1(-1 - b^2 + w_1^2 - w_2^2)1 = 0.43 + 0.1(-1 - 0.43 + 0.43 - 0.23) = 0.307$
So the third update is $\mathbf{w}^3 = (0.553, 0.107, 1.19)$ and $b^3 = 0.307$.

$w_1^4 = w_1^3 + 0.1(-2 - b^3 - w_1^3 - 2w_2^3 + 4w_3^3)1 = 0.553 + 0.1(-2 - 0.307 - 0.553 - 0.214 + 4.76) = 0.7216$
$w_2^4 = w_2^3 + 0.1(-2 - b^3 - w_1^3 - 2w_2^3 + 4w_3^3)2 = 0.107 + 0.2(-2 - 0.307 - 0.553 - 0.214 + 4.76) = 0.4442$
$w_3^4 = w_3^3 + 0.1(-2 - b^3 - w_1^3 - 2w_2^3 + 4w_3^3)(-4) = 1.19 - 0.4(-2 - 0.307 - 0.553 - 0.214 + 4.76) = 0.5156$
$b^4 = b^3 + 0.1(-2 - b^3 - w_1^3 - 2w_2^3 + 4w_3^3)1 = 0.307 + 0.1(-2 - 0.307 - 0.553 - 0.214 + 4.76) = 0.4756$
So the forth update is $\mathbf{w}^4 = (0.7126, 0.4442, 0.5156)$ and $b^4 = 0.4756$.

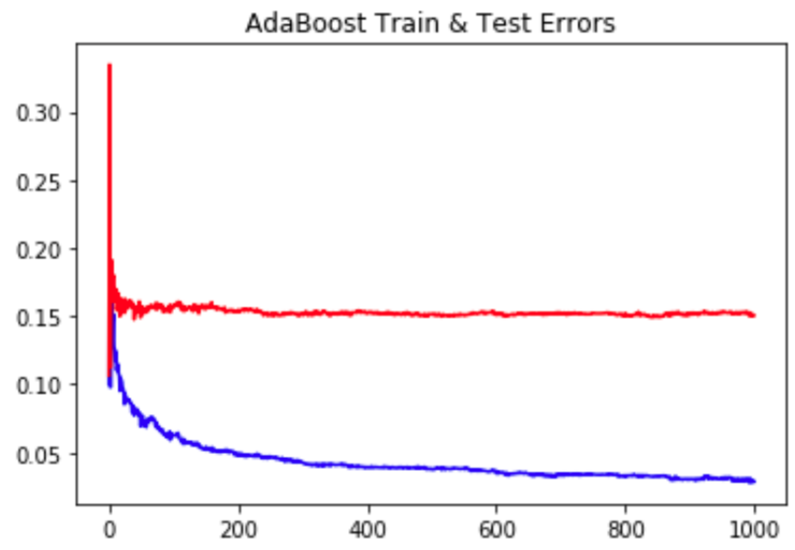
$w_1^5 = w_1^4 + 0.1(0 - b^4 - 3w_1^4 + w_2^4 + w_3^4)3 = 0.7216 + 0.3(0 - 0.4756 - 2.1648 + 0.4442 + 0.5156) = 0.21742$
$w_2^5 = w_2^4 + 0.1(0 - b^4 - 3w_1^4 + w_2^4 + w_3^4)(-1) = 0.4442 - 0.1(0 - 0.4756 - 1.3326 + 0.4442 + 0.5156) = 0.61226$
$w_3^5 = w_3^4 + 0.1(0 - b^4 - 3w_1^4 + w_2^4 + w_3^4)(-1) = 0.5156 - 0.1(0 - 0.4756 - 1.3326 + 0.4442 + 0.5156) = 0.68366$
$b^5 = b^4 + 0.1(0 - b^4 - 3w_1^4 + w_2^4 + w_3^4)1 = 0.4756 + 0.1(0 - 0.4756 - 1.3326 + 0.4442 + 0.5156) = 0.30754$
So the last update is $\mathbf{w}^5 = (0.21742, 0.61226, 0.68366)$ and $b^5 = 0.30754$.

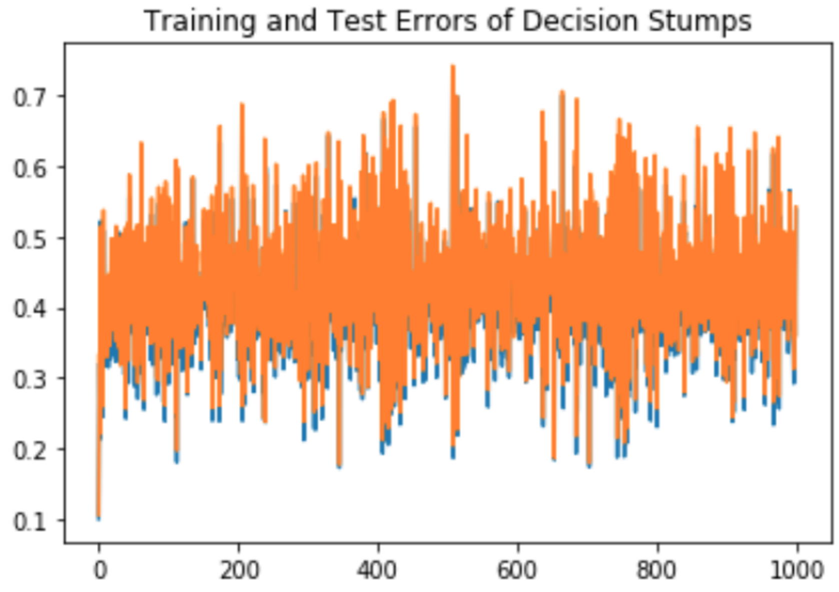
2 Practice [60 points + 10 bonus]

1. I have updated. Please see: `https://github.com/aghababa/pourmahmood`

2. (a) According to the figures, although training and test errors of weak learners (decision stumps) oscillates (of course, within a particular range, say, roughly between 0.3 and 0.8), training and test errors of AdaBoost converges to a same number, approximately 0.18. Having said that, we observe that the general trend for training error is decreasing at first and then being roughly stable. However, test error firstly increases and then remains approximately stable. Also, as the second figure shows, training error of weak learners almost all the time is better than test error, as the blue one is hidden behind the orange one.

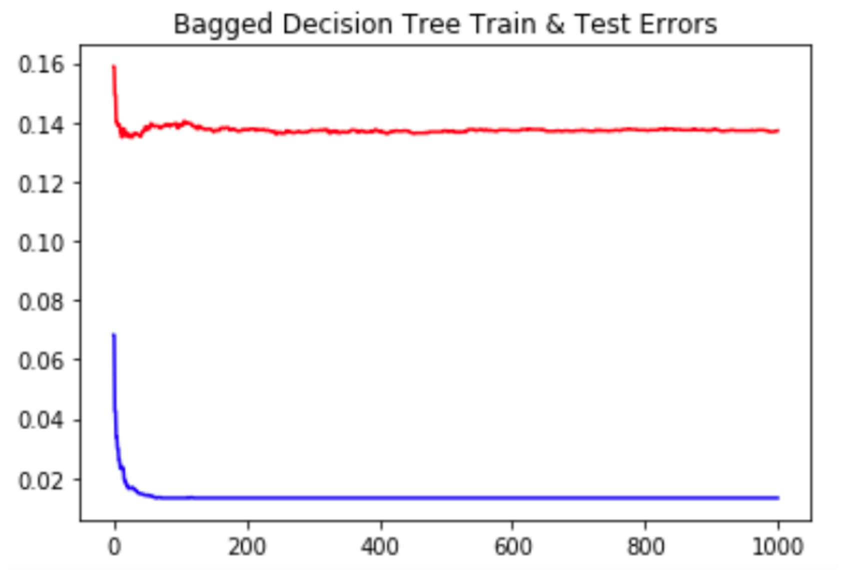
   In the following figures, the blue one is regarding train error and the orange one is regarding test error.





Comparing with the results in decision tree in HW1, we observe that trends are the same. In fact, training error decreases (when we increase the depth of tree in decision tree and the number of trees in bagged tree) but test error increases there. Nevertheless, the numbers are different in errors. In HW1, errors of training data is approximately 0.018 but for bagging is 0.18. For test error in HW1 we got about 0.16 but here about 0.18.

(b) The blue curve is for training data error and the red one is for test data error.



Comparing with decision trees, we can say that bagged trees are better than a single tree. Indeed, it is more observable in test errors. By incrementing the number of trees test errors decline in a general trend, but in decision tree, for instance in HW1 results, test errors increase. Over training examples, it seems they act roughly similarly (both are going down).

According to error figures of bagged trees and Adaboost, they have a relatively similar performance on both training and test data sets. The only difference appears to be the lower percentage of train error in bagged trees when there are a few number of weak learners (a.e. at the beginning of x-axis).

(c) For 100 single decision tree learners the results are as follows:

Bias: 0.35061800000000254
Variance: 0.34134199999999715
Bias + Variance: 0.6919599999999997

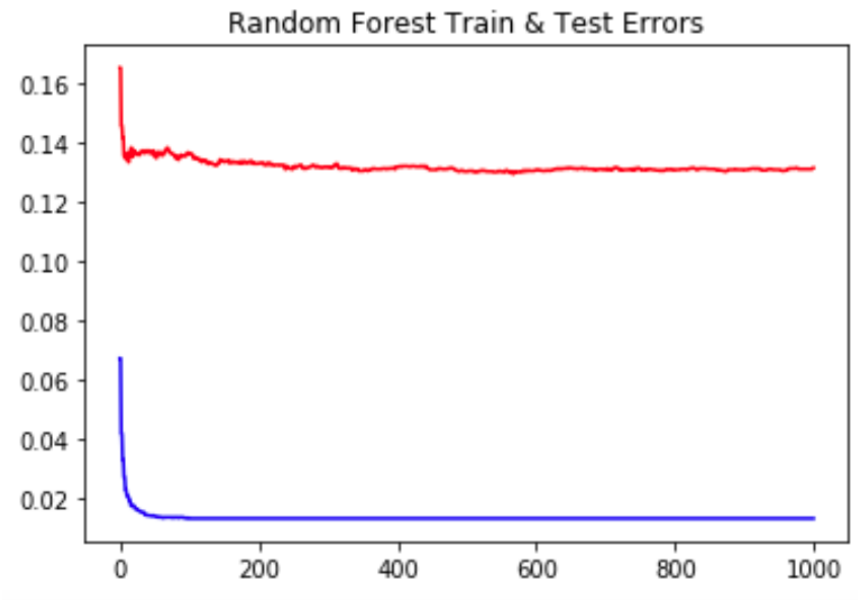
For bagged predictors the results are as follows (Note that as it took a lot of time to run this part of algorithm, I couldn't get the result for 200 bagged predictors each with 1000 trees. Instead, I ran my algorithm with 200 bagged predictors each with 100 trees.):

Bias: 0.3524180000000025
Variance: 0.20194200000000023
Bias + Variance: 0.5543600000000027

Comparing the results, we see that bias is almost the same in both where as variance is decreased. The general squared error 0.691 in 100 single decision tree is reduced to 0.554 in 100 bagged predictors.

According to lecture notes, the reason is that a decision tree tends to overfit and so it has a small bias, but large variance. However, bagging averages a set of decision trees, to maintains similar bias but largely reduces variance.

(d) The blue curve is for training data and the red one is for test data.



Looking at figures of errors in both, we can conclude that they have a roughly similar performance (in this dataset). However, in general, it is possible to have datasets that one of the algorithms perform better than the other.

(e) For a single random tree I got:
Bias = 0.32665781156228874
Variance = 0.45603744748946656.

For whole forest I got:
Bias = 0.19145781156228875
Variance = 0.3309421884377112.

If we compare the bias and variance of a single random tree with a single tree predictor in part (c), we see that they are almost the same.

However, comparing the bias and variance of a bagged tree and a forest we observe that bias is decreased in forest but variance is increased relative to the bagged tree. But the sum of bias and variance remained roughly the same i.e. trade off is hold.

Conclusion is that if in a data we would like to have a lower variance, it is better to use bagged tree algorithm to model the problem. By contrast, if we are planning to have a small bias, it is better to opt for a random forest as a model.

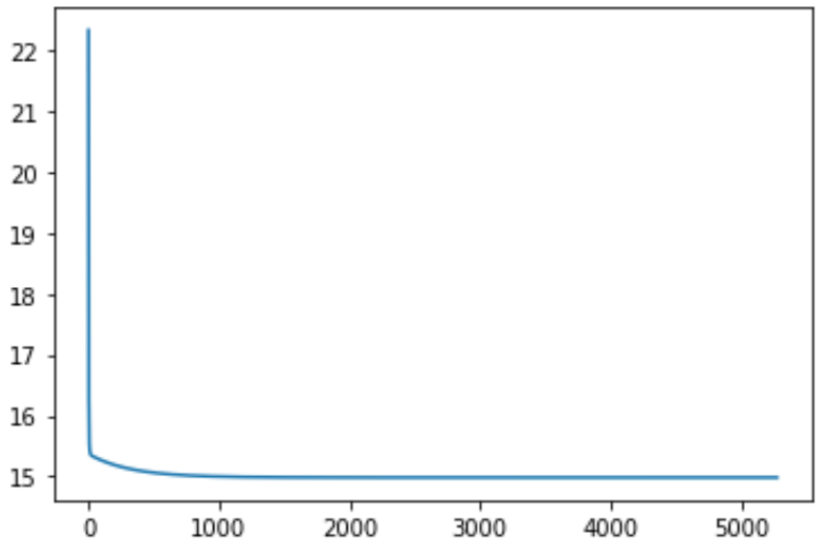
3. [**Bonus**]

4. (a) The learning rate I have chosen is $r = 0.014$ and the weight vector became

$$\mathbf{w}_{\text{batch}} = [-0.01520163, 0.9003219, 0.78604328, 0.85077298, 1.29870037, 0.12985012, 1.57192298, 0.99844599].$$

Using this vector the cost function value of the test data became 23.361436319750908.

Cost function of training data is drawn below:

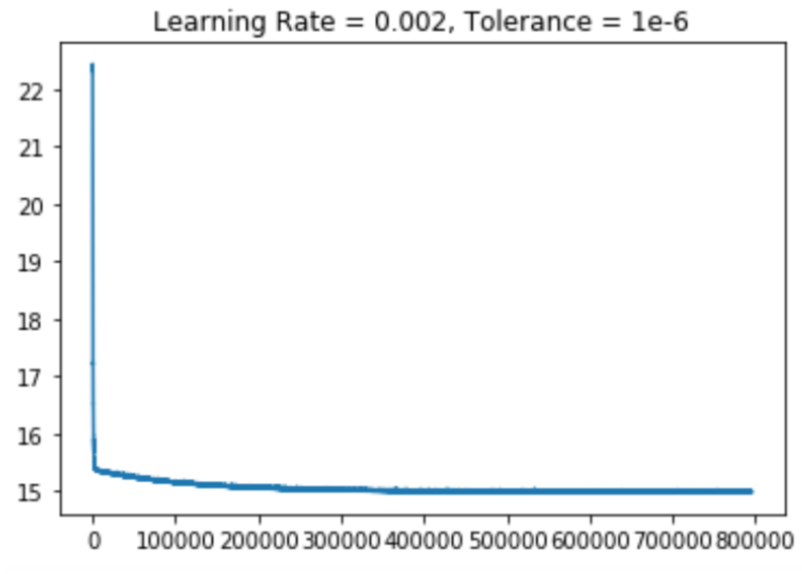


(b) The learning rate I have chosen is $r = 0.002$ and the weight vector became

$$\mathbf{w}_{\text{Stochastic}} = [-0.01656469, 0.86777169, 0.75181352, 0.81410938, 1.27388667, 0.1254294, 1.52788089, 0.96869219].$$

Using this vector the cost function value of the test data became 23.260693434471587.

Cost function of training data is drawn below:



(c) The optimal weight vector in analytical way is:

$$\mathbf{w}_{\min} = [-0.01519667, 0.90056451, 0.78629331, 0.85104314, 1.29889413, 0.12989067, 1.57224887, 0.99869359].$$

Comparing with the weight vector learned by batch gradient descent we see that they are roughly the same. Indeed, their normalized Euclidean distance is near zero, namely,

$$\left\| \frac{\mathbf{w}_{\min}}{\|\mathbf{w}_{\min}\|} - \frac{\mathbf{w}_{\text{batch}}}{\|\mathbf{w}_{\text{batch}}\|} \right\|_2 = 0.00008063467498835986.$$

Comparing with the weight vector learned by stochastic gradient descent we see that they are close together. In fact, their normalized Euclidean distance is:

$$\left\| \frac{\mathbf{w}_{\min}}{\|\mathbf{w}_{\min}\|} - \frac{\mathbf{w}_{\text{Stochastic}}}{\|\mathbf{w}_{\text{Stochastic}}\|} \right\|_2 = 0.008416070066266499.$$

As a consequence, the batch gradient descent approximation for $\mathbf{w}_{\min}$ is better and achieves faster than stochastic gradient descent. The reason is in this example, the size of training data and test data is small and so the batch gradient descent algorithm works well and gets the result faster than the stochastic gradient descent algorithm. However, we know that in big data the stochastic gradient descent algorithm is much faster than the batch gradient descent algorithm.