Problems

1. Logistic and Find test error MATLAB implementation

Part 1

In the Logistic function file, the for loop terminates either iteration reaches the max_its or the gradient goes below the tolerance, in this setting is 10^{-3}. And here's the report,

Table 1: E_{in} and classification error with $10^{-3} tolerance$

Maximum Iterations	10,000	100,000	1,000,000
Number of Iterations	10,000	100,000	1,000,000
E_{in}	0.5847	0.4937	0.4352
E_{class_train}	0.3092	0.2237	0.1513
E_{class_test}	0.3172	0.2069	0.1310

E_{test} is generally lower than E_{in} because E_{in} is an estimation bounded by a loose upper bound, so the actual E_{test} can be lower than the error estimation in training. Moreover, if the gradient isn't below the tolerance, more iterations basically bring the result of lower error rate.

Part 2

The best result is when the maximum iteration is 1 million time, the E_{test}= 0.1310 and it takes 43.50 seconds to achieve the goal. By using `glmfit` function, the E_{test} = 0.1103 and takes only 0.0015 seconds.

Part 3

Scaling the features by subtracting the the mean and dividing by the standard deviation for each of the features, $Z_{train} = \frac{X_{train} - \overline{X}_{train}}{\sigma_{train}}$ and $Z_{test} = \frac{X_{test} - \overline{X}_{test}}{\sigma_{test}}$ By setting a lower tolerance rate, means the number of iterations would decrease. Also, if the learning rate η is larger, it means the function would take a huge step each time. Here's the table that, with a fixed 10^{-6} tolerance, changing the learning rate from 10^{-6} to 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}.

Table 2: E_{in} and E_{test} with $10^{-6} tolerance$ and different learning rate

η	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	3	5
Number of Iterations	23,371,190	2,337,115	233,707	23,367	2,332	229	73	41
E_{in}	0.4074	0.4074	0.4074	0.4074	0.4074	0.4074	0.4074	0.4074
E_{test}	0.1034	0.1034	0.1034	0.1034	0.1034	0.1034	0.1034	0.1034
Execution_Time (sec)	1091	102	10.23	1.0127	0.1016	0.0104	0.0031	0.0017

I'm kind of surprise that, even if the learning rate is pretty large, compare to the previous experiment, with a lower tolerance rate, we can achieve a great E_{in} and E_{test}. with fewer iterations and fewer execution time.

Problems

2. LFD Problem 2.22

First, rewrite the equation,

$$\begin{aligned} E_D[E_{out}(g^{(D)})] =& E_D[E_{x,y}[(g^D(x) - y(x))^2]] \\ =& Ex, y[E_D[(g^D(x) - y(x))^2]] \\ =& E_{x,y}[E_D[(g^D(x)]^2 - 2\overline{g}(x)y(x) + (y(x))^2] \end{aligned} \tag{1}$$

Then we continue to modify the equation,

$$\begin{aligned} E_D[(g^D(x)]^2 - 2\overline{g}(x)y(x) + (y(x))^2 =& E_D[(g^D(x)]^2 - 2\overline{g}(x)(f(x) + \epsilon) + (f(x) + \epsilon)^2 \\ =& E_D[(g^D(x)]^2 - \overline{g}(x)^2 + (\overline{g}(x)^2 - 2\overline{g}(x) + f(x)^2) + \epsilon^2 - 2(\overline{g}(x) + f(x))\epsilon \\ =& E_D[(g^D(x) - \overline{g}(x))^2] + (\overline{g}(x) - f(x))^2 + \epsilon^2 - 2\epsilon(\overline{g}(x) - f(x)) \\ =& var(x) + bias(x) + \epsilon^2 - 2\epsilon(\overline{g}(x) - f(x)) \end{aligned} \tag{2}$$

Then we can put $E_{x,y}$ back to the equation,

$$\begin{aligned} E_D[Eout(g^D)] =& E_{x,y}[var(x)] + E_{x,y}[bias(x)] + E_{x,y}[\epsilon^2] - 2E_{x,y}[(\epsilon(\overline{g}(x) - f(x)))] \\ =& var + bias + E_x[E_{y|x}[\epsilon^2|x]] - 2E_x[E_{y|x}[\epsilon(\overline{g}(x) - f(x))|x]] \\ =& var + bias + E_x[E_\epsilon[\epsilon^2]] - 2E_x[\overline{g}(x) - f(x)E_{y|x}[\epsilon|x]] \\ =& var + bias + E_\epsilon[(\epsilon - E_\epsilon[\epsilon])^2] - 2E_x[(\overline{g}(x) - f(x))E_\epsilon[\epsilon]] \\ =& var + bias + \sigma^2 \end{aligned} \tag{3}$$

Problems

3. LFD Problem 2.24

(a)

$$\begin{aligned}\overline{g}(x) =& E_D[g(x)] \\ =& E_D\left[\frac{y_1 - y_2}{x_1 - x_2}x + \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}\right] \\ =& \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1}\frac{x_1^2 - x_2^2}{x_1 - x_2}dx_1 dx_2 \cdot x + \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1}\frac{x_1 x_2^2 - x_2 x_1^2}{x_1 - x_2}dx_1 dx_2 \\ =& \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1}(x_1 + x_2)dx_1 dx_2 \cdot x - \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1}(x_1 x_2)dx_1 dx_2 \\ =& 0\end{aligned} \tag{4}$$

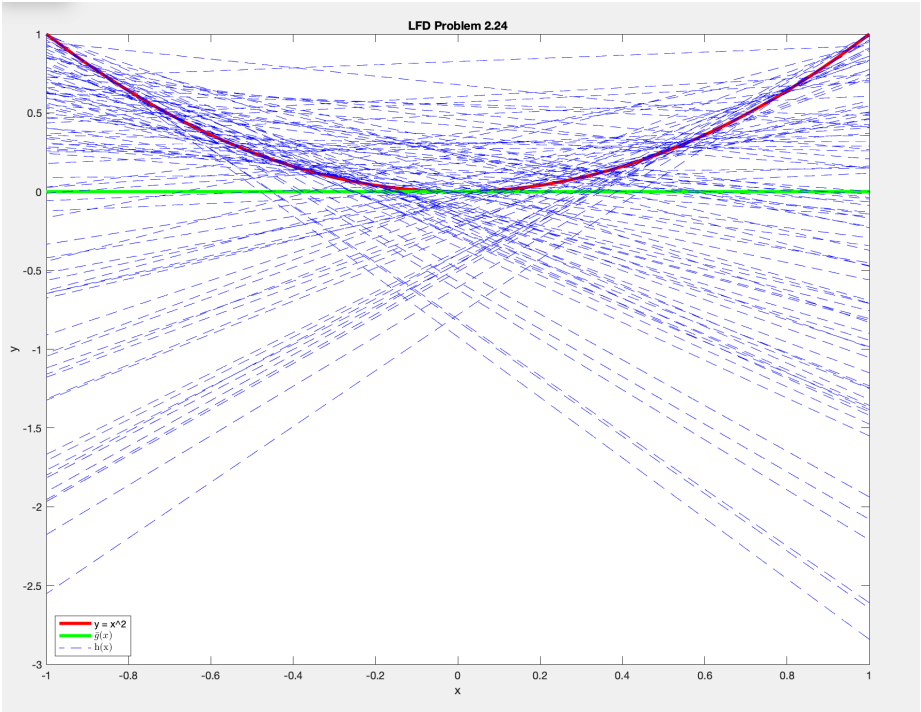
(b)

We can simulate the process for, let's say 100 times, determining $\overline{g}(x)$, $E[E_{out}]$, *bias*, and *var*. The code on MATLAB is like

```
x = -1:0.00001:1;
y = x.^2;
z = 0 * x;
bias = 0;
Eout2 = 0.0;
var2 = 0.0;
for i = 1:10000
    var1 = 0;
    Eout1 = 0;
    rand1(i) = randperm(length(x),1);
    rand2(i) = randperm(length(x),1);
    randx1(i) = x(rand1(i));
    randx2(i) = x(rand2(i));
    randy1(i) = y(rand1(i));
    randy2(i) = y(rand2(i));
    m(i) = (randy1(i) - randy2(i))/(randx1(i) - randx2(i));
    b(i) = (randx1(i)*randy2(i) - randx2(i)*randy1(i))/(randx1(i) - randx2(i));
    for j = (1:length(x))
        if(x(j) ~= randx1(i))
            if(x(j) ~= randx2(i))
                var1 = var1 + (m(i)*x(j)+b(i) - 0)^2;
                Eout1 = Eout1 + (m(i)*x(j)+b(i) - y(j))^2;
            end
        end
    end
    var2 = (var1/199999) + var2;
    Eout2 = (Eout1/199999) + Eout2;
end
varAvg = var2/10000;
varRound = round(varAvg,2);
Eout = Eout2/10000;
EoutRound = round(Eout, 2);

for i = 1:200001
    bias = bias + (0 - y(i))^2;
```

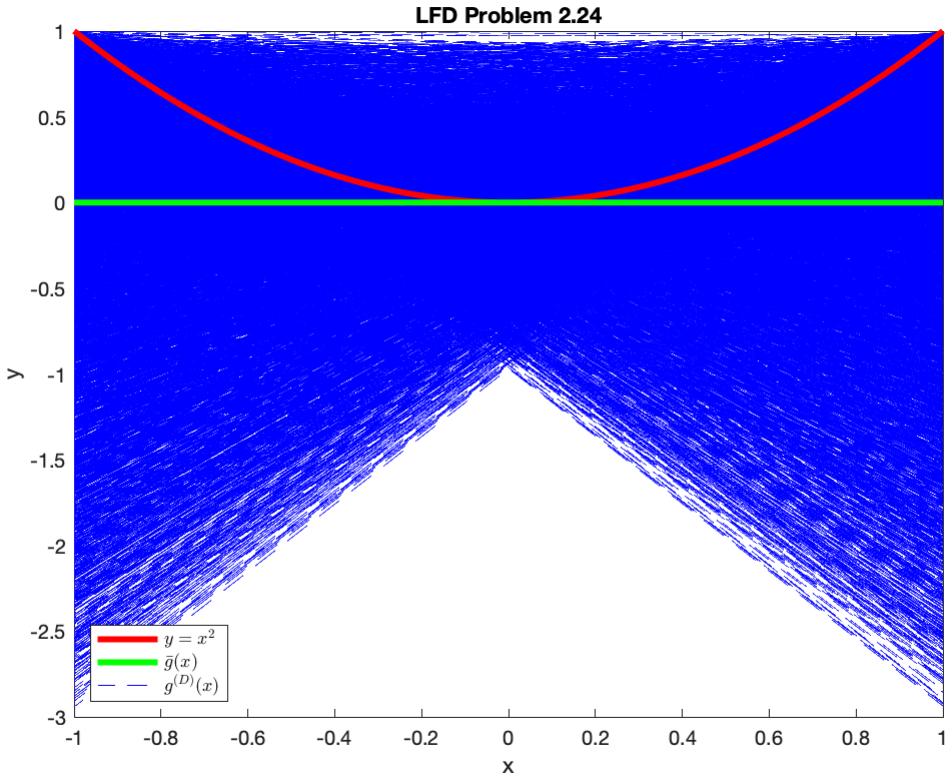
```
end
biasAvg = bias/200001;
biasRound = round(biasAvg,2);
E_Eout = biasRound+varRound;

g_x = plot(x, m(1)*x + b(1), 'b--', 'LineWidth', 0.2)
hold on
for i = 2:10000
    plot(x, m(i)*x + b(i), 'b--', 'LineWidth', 0.2)
end
target_func = plot(x,y, 'r', 'LineWidth', 3)
gbar_func = plot(x,z, 'g', 'LineWidth', 3)
hold off
title('LFD Problem 2.24')
xlabel('x')
ylabel('y')
legend([target_func, gbar_func, g_x],'$y = x^2$','$\bar{g}(x)$','$g^{(D)}(x)$',
    'Interpreter', 'latex', 'Location', 'SouthWest')
hold off
```

And the graph for the simulation is like,



(c)

Based on my simulation code, the length of x and y are 200001, and we sample 2 data points out of it for 10000 times. Each time we can formulate a linear hypothesis for predicting y. The result is showing below,



E_{out} is the actual out-of-sample error, E_E_{out} means expected value of out-of-sample error which is calculated by sum of bias and variance. The result represents that $E_{out} = bias + var$

(d)

First, we compute E_{out}

$$\begin{aligned} E_{out} = E[(g(x) - f(x))^2] =& E_x[(ax + b - x^2)^2] \\ =& E_x[x^4] - 2aE_x[x^3] + (a^2 - 2b)E_x[x^2] + 2abE_x[x] + b^2 \\ =& \frac{1}{2}\int_{-1}^{1} x^4 dx - 2a\frac{1}{2}\int_{-1}^{1} x^3 dx + (a^2 - 2b)\frac{1}{2}\int_{-1}^{1} x^2 dx + 2ab\frac{1}{2}\int_{-1}^{1} x dx + b^2 \\ =& \frac{1}{5} + \frac{(a^2 - 2b)}{3} + b^2 \end{aligned} \tag{5}$$

Then, we can take expectation with respect to D to get the test performance, and replace a and b with $\frac{(y_1 - y_2)}{(x_1 - x_2)}$ and $\frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}$ respectively, and, again, replace y_1 and y_2 with x_1^2 and x_2^2, we get,

$$\begin{aligned} E_D[E_{out}] =& \frac{1}{5} + \frac{1}{3}E_D[(x_1 + x_2)^2 + 2x_1x_2] + E_D[x_1^2 x_2^2] \\ =& \frac{1}{5} + \frac{1}{3}\cdot\frac{1}{4}\int_{-1}^{1}\int_{-1}^{1}[x_1^2 + x_2^2 + 4x_1x_2]dx_1dx_2 + \frac{1}{4}[x_1^2x_2^2]dx_1dx_2 \\ =& \frac{1}{5} + \frac{1}{3}\cdot\frac{1}{4}\cdot\frac{8}{3} + \frac{1}{4}\cdot\frac{4}{9} = \frac{8}{15} \approx 0.53 \end{aligned} \tag{6}$$

The we can compute the bias and variance,

$$bias(x) = (\overline{g}(x) - f(x))^2 = x^4 = E[x^4] = \frac{1}{2}\int_{-1}^{1} x^4 dx = \frac{1}{5}$$

$$\begin{aligned} var(x) =& E_D[(g(x) - \overline{g}(x))^2] = E_D[a^2x^2 + 2abx + b^2] \\ =& E_D[a^2]x^2 + 2E_D[ab]x + E_D[b^2] \\ =& E_D[(x_1 + x_2)^2]x^2 - 2E_D[(x_1 + x_2)x_1x_2]x + E_D[x_1^2x_2^2] \\ =& \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1}(x_1^2 2x_1x_2 + x_2^2)dx_1dx_2 \cdot x^2 - \frac{1}{2}\int_{-1}^{1}\int_{-1}^{1}(x_1^2x_2 + x_1x_2^2)dx_1dx_2 \cdot x + \frac{1}{4}\int_{-1}^{1}\int_{-1}^{1} x_1^2x_2^2 dx_1dx_2 \\ =& \frac{1}{4}(\frac{4}{3} + 0 + \frac{4}{3}) \cdot x^2 - 0 \cdot x + \frac{1}{4}\frac{4}{9} = \frac{2}{3} \cdot x^2 + \frac{1}{9} \end{aligned} \tag{7}$$

then take the expectation with respect to x,

$$var = E_x[\frac{2}{3}x^2 + \frac{1}{9}] = \frac{2}{3}\frac{1}{2}\int_{-1}^{1} x^2 dx + \frac{1}{9} = \frac{1}{3}$$

$bias + var = 0.2 + \frac{1}{3} \approx 0.53$

Problems

4. LFD Exercise 3.4

(a)

Based on what we know, $y = Xw^* + \epsilon$ from textbook,

$$\begin{aligned} y =& Xw^* + \epsilon \\ \hat{y} =& Hy \\ \hat{y} =& H(Xw^* + \epsilon) \\ \hat{y} =& (X(X^TX)^{-1}X^T)Xw^* + H\epsilon \\ \hat{y} =& (XX^{-1}(X^T)^{-1}X^T)Xw^* + H\epsilon \\ \hat{y} =& (II)Xw^* + H\epsilon \\ \hat{y} =& Xw^* + H\epsilon \end{aligned} \tag{8}$$

(b)

Simply replace $\hat{y}$ and y with equation as below,

$$\begin{aligned} y =& Xw^* + \epsilon \\ \hat{y} =& Xw^* + H\epsilon \\ \hat{y} - y =& Xw^* + H\epsilon - (Xw^* + \epsilon) \\ =& (H - I)\epsilon \end{aligned} \tag{9}$$

where I is an identity matrix.

(c)

Based on part(b) and the $E_{in}[W_{[}lin]] = \frac{1}{N}||\hat{y} - y||^2$, we can rewrite the equation,

$$\begin{aligned} E_{in}[W_{lin}] =& \frac{1}{N}||\hat{y} - y||^2 \\ =& \frac{1}{N}||(H - I)\epsilon||^2, \text{using the fact from part(b)} \\ =& \frac{1}{N}(\epsilon^T H^T H\epsilon - 2\epsilon^T H^T\epsilon + \epsilon^T I^T I\epsilon), \text{matrix algebra} \\ =& \frac{1}{N}(\epsilon^T H\epsilon - 2\epsilon^T H\epsilon + \epsilon^T\epsilon), \text{Using the fact that H are symmetric and } H^2 = H \\ =& \frac{1}{N}(-\epsilon^T H\epsilon + \epsilon^T I\epsilon) \\ =& \frac{1}{N}(\epsilon^T(I - H)\epsilon) \\ =& \frac{1}{N}\epsilon^T\epsilon - \frac{1}{N}\epsilon^T H\epsilon \end{aligned} \tag{10}$$

Problems

5. LFD Problem 3.4

(a)

$$e_n(w) = \begin{cases} 0 & \text{if } y_n w^T x_n > 1 \\ (1 - y_n w^T x_n)^2 & \text{if } y_n w^T x_n < 1 \end{cases}$$

and now we have,

$$\lim_{w: y_n w^T x_n \to \pm 1} e_n(w) = 0$$

which is the definition of continuous function.

Then we want to prove it's differentiable,

$$\nabla e_n(w) = \begin{cases} 0 & \text{if } y_n w^T x_n > 1 \\ -2y_n(1 - y_n w^T x_n)x_n & \text{if } y_n w^T x_n < 1 \end{cases}$$

and we have,

$$\lim_{w: y_n w^T x_n \to \pm 1} \nabla e_n(w) = 0$$

shows that it's differentiable as well.

(b)

Now we list two functions we're interested in,

$$sign(w^T x_n) \neq y_n = \begin{cases} 0 & \text{if } sign(w^T x_n) = y_n \\ 1 & \text{if } sign(w^T x_n) \neq y_n \end{cases}$$

$$e_n(w) = \begin{cases} 0 & \text{if } y_n w^T x_n > 1 \\ (1 - y_n w^T x_n)^2 & \text{if } y_n w^T x_n < 1 \end{cases}$$

Imagine if $sign(w^T x) \neq y_n$, which means the equation $sign(w^T x_n) \neq y_n) = 1$, and since the sign of $w^T x_n$ is not equal to the sign of y_n, for the equation $e_n(w) = (max(0, 1 - y_n w^T x_n))^2$, the term $-y_n w^T x_n$ now is positive. So the overall $1 - y_n w^T x_n$ is greater than 1. So the claim holds in this case.

Here's the mathematic expression,

$$\begin{aligned} &sign(w^T x_n) \neq y_n \rightarrow (sign(w^T x_n) \neq y_n) = 1 \\ &-y_n w^T x_n > 0 \rightarrow e_n(w) = (max(0, 1 - y_n w^T x_n))^2 > 1 \end{aligned} \tag{11}$$

The other case, when $sign(w^T x_n) = y_n$, $(sign(w^T x_n) \neq y_n) = 0$, and since the sign of $w^T x_n$ is equal to y_n, for the equation $e_n(w) = (max(0, 1 - y_n w^T x_n))^2$, the minimum value is 0, so the claim holds as well.

Here's the mathematic expression,

$$\begin{aligned} sign(w^T x_n) = y_n \rightarrow (sign(w^T x_n) \neq y_n) = 0 \\ y_n w^T x_n < 0 \rightarrow e_n(w) = (max(0, 1 - y_n w^T x_n))^2 \geq 0 \end{aligned} \tag{12}$$

Since for each n, $e_n(w) \geq (sign(w^T x_n) \neq y_n)$, we can say e_n is an upper bound for $sign(w^T x_n) \neq y_n$, then we can have the function below,

$$\frac{1}{N} \sum_{n=1}^{N} e_n(w) \geq \frac{1}{N} \sum_{n=1}^{N} (sign(w^T x_n) \neq y_n) = E_{in}(w)$$

It's proved.

(c)

If we apply stochastic gradient descent on $\frac{1}{N} \sum_{n=1}^{N} E_n(w)$, we can get the following algorithm.

First, Select an initial w,

Second, repeat until condition: select (x_n, y_n) randomly and let $s_n = w^T x_n$. If $y_n s_n \leq 1$, means s_n is further from y_n so we do the following

$$\nabla e_n(w) = -2y_n(1 - y_n w^T x_n)x_n$$

and then update w as

$$w(t+1) = w(t) + \eta \nabla e_n(w) = w(t) + 2\eta y_n(1 - y_n s_n)x_n = w(t) + \eta'(y_n - s_n)x_n$$

If $y_n s_n > 1$ means s_n does well, and since $y_n s_n > 1$ will lead $e_n(w) = 0$, the $\nabla e_n(w)$ is also 0, then,

$$w(t+1) = w(t) + \eta \cdot 0 = w(t)$$

, which means the algorithm does nothing.

This whole algorithm is exactly Adaline algorithm.

Problems

6. LFD Problem 3.19

(a)

Let's say we have 3 data points, x_1, x_2, and x_3, based on the transformer, if all these three data points are different pairwise, then x_1 would be transformed to $(1, 0, 0)$.

We can imagine that, for this transformation, one potential problem is the complexity goes to infinity as n grows large. The other problem is, although it's not unusual to map each point not in the data set to 0, this is still a waste as n grows.

(b)

Using the example in part(a), we have 3 data points.

For this new transformer, and let's say we want to transform x_2, it becomes,

$$\Phi(x_2) = \begin{bmatrix} e^{(-\frac{||x_2-x_1||^2}{2\gamma^2})} \\ e^{(-\frac{||x_2-x_2||^2}{2\gamma^2})} \\ e^{(-\frac{||x_2-x_3||^2}{2\gamma^2})} \end{bmatrix} = \begin{bmatrix} \phi_1(x_2) \\ \phi_2(x_2) \\ \phi_3(x_2) \end{bmatrix}$$

In this transformer, when our number of data points increase, the dimensionality also increases. So this may be a problem if we have infinite number of data, the space complexity for the transformer would be pretty complicated.

(c)

For this transformer, the potential problem is that if we only have 3 data points, then this transformer is quite a waste since it creates a matrix with size of 101×101. However, compared with the previous 2 transformers, this one is much better since we know the dimensionality is fixed, though might be a waste, and as the size of training data grows, we can still have decent space complexity that is not corresponded to the number of data we have.