417 hw2

alex.huang

February 2020

1

1.1

When the number of iterations is 10^4, we have $E_{in} = 0.5847$, and binary classification error being 0.3092 and 0.3172 on training and testing data, respectfully. The training took the totality of 0.705689 seconds.

When the number of iterations is 10^5, we have $E_{in} = 0.4937$, and binary classification error being 0.2237 and 0.2069 on training and testing data, respectfully. The training took the totality of 6.961933 seconds.

When the number of iterations is 10^6, we have $E_{in} = 0.4354$, and binary classification error being 0.1513 and 0.1310 on training and testing data, respectfully. The training took the totality of 68.596270 seconds.

By the numbers from the training, we can see that as the maximum number of iterations grow, the training takes far longer time, yet all the three errors: E_{in} and two binary classification errors decrease significantly. Therefore, we can conclude that training with more iterations improve the model and decrease error, in this case.

For all three cases, the binary classification error of the testing and training data are similar, with small differences. We can conclude that the model works well, since the generalization error is small, and we are relatively confident that the out of sample error is similar to those errors. The simplicity of our model might be one of the reasons.

1.2

In this part, using glmfit we have generated a result with $E_{in} = 0.4074$, with the binary classification errors on training and testing data calculated as 0.1711 and 0.1103. The total time is 0.032136 seconds.

1.3

When the eta is 0.01, we have $E_{in} = 0.4074$, and binary classification error being 0.1034 on testing data, respectfully. The training took 23368 iterations to terminate. The training took the totality of 1.521945 seconds.

When the eta is 0.1, we have $E_{in} = 0.4074$, and binary classification error being 0.1034 on testing data, respectfully. The training took 2333 iterations to terminate. The training took the totality of 0.262555 seconds.

When the eta is 1, we have $E_{in} = 0.4074$, and binary classification error being 0.1034 on testing data, respectfully. The training took 230 iterations to terminate. The training took the totality of 0.035201 seconds.

When the eta is 4, we have $E_{in} = 0.4074$, and binary classification error being 0.1034 on testing data, respectfully. The training took 54 iterations to terminate. The training took the totality of 0.016343 seconds.

When the eta is 6, we have $E_{in} = 0.4074$, and binary classification error being 0.1034 on testing data, respectfully. The training took 32 iterations to terminate. The training took the totality of 0.010661 seconds.

When the eta is 7, we have $E_{in} = 0.4074$, and binary classification error being 0.1034 on testing data, respectfully. The training took 43 iterations to terminate. The training took the totality of 0.013334 seconds.

Normalizing the data is definitely helpful. We can see that after normalizing the data, the training takes relatively small amount of time, but resulted, in all cases, a smaller E_{in} and binary classification error. It is a smart step to normalize the data.

If eta is increased, first we see a huge drop in learning time and iterations, probably because we now take fewer steps to reach to optimal solution. However, we can see that as eta goes from 6 to 7, it actually took more iterations because we have reached a point where 6 is efficient enough, and larger eta would have a negative effect. Also, we can see that increasing eta only has effects on iterations, but not on the errors measured.

2

By what we've proven in class, we know that

$$E_D[E_{out}(g^D)] = E_x[E_D[(g^D(x) - \bar{g}(x))^2]] + E_x[(\bar{g}(x) - y(x))^2] = var + E_x[(\bar{g}(x) - (f(x) + \epsilon))^2]$$

Furthermore, we can see that

$$E_x[(\bar{g}(x) - (f(x) + \epsilon))^2] = E_x[\bar{g}(x)^2] + E_x[(f(x) + \epsilon)^2] - 2E_x[\bar{g}(x)(f(x) + \epsilon)]$$

$$E_x[(f(x) + \epsilon)^2] = E_x[f(x)^2] + 2E_x[f(x)\epsilon] + E_x[\epsilon^2]$$

$$E_x[g(x)(f(x) + \epsilon)] = E_x[g(x)f(x)] + E_x[g(x)\epsilon]$$

Note that since ϵ is a zero mean random variable with variance ϵ^2, (also, $\sigma^2 = var(\epsilon) = E[\epsilon^2] - E[\epsilon]^2 = E[\epsilon^2]$), we know that $E_x[(f(x) + \epsilon)^2] = E_x[f(x)^2] + var$, and $E_x[g(x)(f(x) + \epsilon)] = E_x[g(x)f(x)]$

Also, because

$$bias = E_x[(g(x) - f(x))^2] = E_x[g(x)^2] + E_x[f(x)^2] - 2E_x[g(x)f(x)]$$

We can see that $E_x[(g(x) - (f(x) + \epsilon))^2] = \sigma^2 + E_x[(g(x) - f(x))^2] = bias + \sigma^2$. Therefore, we have proven that $E_D[E_{out}(g^D)] = bias + var + \sigma^2$.

3

3.1

For any x_1, x_2 we choose, we can see that the best fit is a line linking (x_1, x_1^2) and (x_2, x_2^2), which we can calculate a (in the function $h = ax + b$) as $a = \frac{x_1^2 - x_2^2}{x_1 - x_2} = x_1 + x_2$. (assuming $x_1 \neq x_2$, which case makes the learning impossible) Furthermore, we know that $b = -x_1 x_2$ by basic analytical geometry. To get $E_D[a]$, we can take the integral for x_1, x_2: $\int_{-1}^{1} \int_{-1}^{1} (x_1 + x_2) dx_1 dx_2 = 0$, similarly, $E_D[b] : \int_{-1}^{1} \int_{-1}^{1} (-x_1 x_2) dx_1 dx_2 = 0$. Therefore, the expected value for the average function, by definition, is $\bar{g}(x) = 0$

3.2

We first randomly generate a pair of data points on the interval, then we use the algorithm to determine the line fitting those two points. That is our result of learning. Repeat this task many times, then we've gotten the average function $\bar{g}(x)$ by taking the average of those lines.

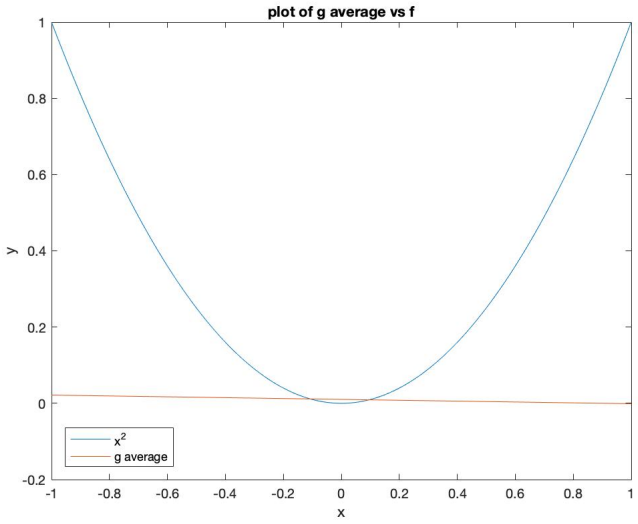
Then we repeat the process of generating data set and learning (denote the resulting line as $ax + b$). For each $g(x)$ generated, pick a large number of evenly distributed data points from -1 to 1, as $x_1, \ldots, x_N$. For each x_i, calculate the following: $(x_i^2 - ax_i - b)^2$, $(x_i^2 - \bar{g}(x_i))^2$, and $(\bar{g}(x_i) - ax_i - b)^2$. Get the average of the three values over $x_1 \ldots, x_N$, and denote them as i, j, k, respectively. Record the i, j, k values for this experiment.

After that, we process the data and calculate the expected values of i, j, k for the experiments, which gets us E_{out}, var and bias, respectively.

3.3

After running the experiment, I got $E_{out} = 0.5292, var = 0.3298, bias = 0.1974$. $bias + var = 0.5272$. Therefore, we can conclude that E_{out} is very close to bias+var.

Below is the graph of g average versus f. Notice that g confirms our theoretical calculations, with negligible deviation since we randomly generated the data set.



3.4

To calculate bias, we need to calculate $E_x[(\bar{g}(x) - f(x))^2]$. Since we already calculated that $\bar{g}(x) = 0$, we only need to compute $E_x[f(x)^2] = E_x[x^4] = \frac{1}{2}\int_{-1}^{1} x^4 dx = 0.2$.

For any training data set (x_1, x_2), we know that the learning result is the function $g^D(x) = (x_1 + x_2)x - x_1x_2$. Therefore, the variance is

$$E_x[E_D[(g^D(x)-\bar{g}(x))^2]] = E_x[E_D[g^D(x)^2]] = E_x[\frac{1}{4}\int_{-1}^{1}\int_{-1}^{1}((x_1+x_2)x-x_1x_2)^2 dx_1 dx_2] = E_x[\frac{1}{9}(6x^2+1)]$$

Then, we have $E_x[\frac{1}{9}(6x^2+1)] = \frac{1}{2}\int_{-1}^{1}\frac{1}{9}(6x^2+1)dx = \frac{1}{3}$. Therefore, the expected variance is $\frac{1}{3}$.

By what we've covered in class, $E_{out} = var + bias = 0.533$

4

4.1

Suppose we have w that satisfies $0 < 1 - y_n w^T x_n$, which means that $E_n = (1 - y_n w^T x_n)^2$. Also, because we know that $1-y_n w^T x_n$ is a continuous, differentiable function (since y_n, x_n are both constant), we know that there is a neighbourhood nearby w such that $0 < 1 - y_n w^T x_n$ is still true. That makes the function continuous and differentiable at w. $E_n = (1 - y_n w^T x_n)^2$, and it is easy to calculate the gradient in this situation: $\nabla E_n(w) = -2y_n x_n + 2x_n(w^T x_n)$ ($y_n^2 = 1$), with k equal to the vector with all 1, the same length as w.

Suppose we have w that satisfies $0 > 1 - y_n w^T x_n$, which means that $E_n = 0$. We know that there is a neighbourhood nearby w such that $0 > 1 - y_n w^T x_n$ is still true. That makes the function continuous and differentiable at w. Therefore, by definition, in this neighbourhood, $E_n = 0$, and therefore $\nabla E_n(w) = 0$.

Suppose that $0 = 1 - y_n w^T x_n$, we now know that $1 = y_n w^T x_n$, and therefore the gradient $\nabla E_n(w) = -2y_n x_n + 2x_n(w^T x_n) = 0$. We can see that at the boundary between 0 and $1 - y_n w^T x_n$, the value and the derivative are the same on both sides. Therefore, on this point we can see that the function is continuous and with a gradient of always 0 on all directions.

Therefore, the function is continuous and differentiable.

4.2

Since by definition $E_n(w) \geq 0$, it is evident that it is an upper bound when $sign(w^T x_n) = y_n$. If $sign \neq y_n$, we know that $y_n w^T x_n \leq 0$, and therefore $1 - y_n w^T x_n \geq 1$, therefore, $E_n(w) \geq 1 = [sign(w^T x_n) \neq y_n]$. This proves the upper bound.

By definition, we know that $E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} [sign(w^T x_n) \neq y_n]$, and by our previous conclusion we have $\frac{1}{N} \sum_{n=1}^{N} E_n(w) \geq \frac{1}{N} \sum_{n=1}^{N} [sign(w^T x_n) \neq y_n] = E_{in}(w)$.

4.3

By definition, we know that stochastic gradient descent follow the following pattern: $w(t+1) \leftarrow w(t) - \eta \nabla_w e_n(w(t))$, with η being a constant. We define $e_n(w)$ as part a, and $s(t) = w(t)^T x(t)$. Suppose $y(t)s(t) > 1$, then we know that $1 - y_n w^T x_n < 0$, which indicates that $e_n(w) = 0$. That corresponds to the decision of not changing $w(t)$.

If $y(t)s(t) \leq 1$, the algorithm updates $w(t)$ by $\eta(y(t) - s(t))x(t) = \eta(y(t) - w(t)^T x(t))x(t)$. We also know that in this case, $e_n(w(t)) = -2y_n x_n + 2x_n(w(t)^T x_n)$, which is exactly twice as the number in the algorithm (we can remove the 2 by changing η). Therefore, in both cases the algorithm in 1.5 corresponds with what we have in $e_n(w)$, so the algorithm does stochastic descent on $\frac{1}{N} \sum_{n=1}^{N} E_n(w)$ (it is stochastic because $E[e_n(w)] = \frac{1}{N} \sum_{n=1}^{N} E_n(w)$).

5

We covered in class that in the VC theory, $E_{out} \leq E_{in} + O(\sqrt{dvc \frac{\ln N}{N}})$, so therefore if our transform resulted in a situation with infinite (not bounded above) VC dimension, our E_{out} can also go out of bound, which is bad.

Also, we have covered in class that in a d-dimensional space, the PLA VC dimension is d+1.

5.1

In this part, if we run the feature transform with N data points, we will transform this thing into a N-dimension space, with $x_1, x_2, \ldots, x_N$ being mapped to $(1, 0, 0 \ldots), (0, 1, 0 \ldots), \ldots$, and all other points mapped to the zero vector. Whatever N is, we will have a space where all the x_n are mapped to linearly independent vectors, which means we will be able to shatter any data set with a linear separator (PLA). The vc dimension is therefore infinity. Therefore, the E_{out} in this case is not bounded, and we cannot guarantee our result can generalize.

5.2

The characteristics of this feature transform is that it maps x to an N dimensional space, with each dimension computed as a function of $\|x - x_n\|$. Also, as γ gets very small, all values will be mapped to a place close to 0, while x_1 will be mapped close to $(1, 0, 0 \ldots)$, and the similar for $x_2, x_3, \ldots$. This transformation is essentially similar to part a. It is easy to see since we are mapped to a N dimensional space, it is always possible that we have a set of size N (no matter how large N gets) and it can be shattered by the learning algorithm. Therefore, the algorithm eventually has a vc dimension of infinity, and as the result, no appropriate bound to E_{out}, and no way to generalize.

5.3

This feature transform generates a huge vector after the transform, because it both i, j has 101 possibilities, and we need to use the combination of them. If the number of data points are small, this is not a great feature transform, since we are being mapped to a huge vector. However, as N goes large, learning is feasible because the dimension is limited, and therefore PLA can have a finite VC dimension. This is guaranteed by what we've covered in class. As PLA has limited VC dimensions, we know that learning is feasible because E_{out} can be bounded.