COMS 4771: Machine Learning (Fall 2018) - Homework #3

Amal Alabdulkarim (aa4235), Danyang He (dh2914), Jing Qian (jq2282)

November 20, 2018

Problem 1

(i)

Given (y, X) and (α, λ), we can augment the data by adding d more "fake data points": $\sqrt{\lambda\alpha}\vec{e_i}$($\vec{e_i} \in R^d$, represents the vector having only i^{th} element nonzero) and augment the label by adding d more 0 labels. Thus, the augmented data matrix $X^* \in R^{d\times(n+d)}$, $y^* \in R^{1\times(n+d)}$.

$$X^* = (1+\alpha_2)^{-\frac{1}{2}}\begin{bmatrix} X & \sqrt{\alpha_2}I \end{bmatrix},\ \ y^* = \begin{bmatrix} y & \mathbf{0} \end{bmatrix}, \gamma = \frac{\alpha_1}{\sqrt{1+\alpha_2}}, w^* = (\sqrt{1+\alpha_2})w \tag{1}$$

$$\alpha_1 = \lambda(1-\alpha), \quad \alpha_2 = \lambda\alpha \tag{2}$$

The elastic net optimization objective function can be written as the following Lasso form[4]:

$$\begin{aligned} L^{Lasso}(\gamma, w) &= ||w^*X^* - y^*||_2^2 + \gamma||w^*||_1 \\ &= ||w^*(1+\alpha_2)^{-\frac{1}{2}}\begin{bmatrix} X & \sqrt{\alpha_2}I \end{bmatrix} - \begin{bmatrix} y & 0 \end{bmatrix}||_2^2 + \gamma||w^*||_1 \\ &= ||w^*(1+\alpha_2)^{-\frac{1}{2}}X - y||_2^2 + \frac{\alpha_2}{1+\alpha_2}||w^*I||_2^2 + \gamma||w^*||_1 \\ &= ||y||_2^2 + \frac{||w^*X||_2^2}{1+\alpha_2} - \frac{2w^*X}{\sqrt{1+\alpha_2}}y^T + \frac{\alpha_2}{1+\alpha_2}||w^*I||_2^2 + \gamma||w^*||_1 \\ &= ||y||_2^2 + ||wX||_2^2 - 2wXy^T + \alpha_2||w||_2^2 + \alpha_1||w||_1 \\ &= ||wX - y||_2^2 + \alpha_2||w||_2^2 + \alpha_1||w||_1 = L^{Elasticnet}(w) \end{aligned} \tag{3}$$

(ii)

Let $y = (y_1, ...y_n)$, X is a $n\times d$ matrix representing the data matrix and $w = (w_1, ..., w_d)$ representing the weight vector. From the description of the problem, we can know each

$y_i \sim (x_i^T w, \sigma^2)$.

$$P(w|y,X) \propto P(y|w,X)P(w) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} exp\{-\frac{(y_i - x_i^T w)^2}{2\sigma^2}\} \prod_{j=1}^{d} \frac{1}{\sqrt{2\pi}\tau} exp(-\frac{w_j^2}{2\tau^2}) \tag{4}$$

Maximizing the posterior probability is equivalent to maximizing the log posterior probability:

$$\ln P(w|y,X) = -\sum_{i=1}^{n} \frac{(y_i - x_i^T w)^2}{2\sigma^2} - \sum_{j=1}^{d} \frac{w_j^2}{2\tau^2} + constant \tag{5}$$

Maximizing the log posterior probability is equivalent to minimizing the negative of equation (5):

$$\begin{aligned} &min \ \sum_{i=1}^{n} \frac{(y_i - x_i^T w)^2}{2\sigma^2} + \sum_{j=1}^{d} \frac{w_j^2}{2\tau^2} \\ \iff &min \ ||y - X^T w||_2^2 + \alpha ||w||_2^2 \ , \alpha = \frac{\sigma^2}{\tau^2} \end{aligned} \tag{6}$$

which is in the form of ridge optimization objective function.

Problem 2

(i)

$$D_{T+1}(i) = \frac{1}{m}\frac{1}{\prod_t Z_t} exp(-y_i g(x_i)) \tag{7}$$

Proof. Starting from the left hands side:

$$\begin{aligned}
D_{T+1}(i) &= \frac{1}{Z_t} D_t(i) exp(-y_i(a_t f_t(x_i))) \\
&= \frac{1}{Z_t}\frac{1}{Z_{t-1}} D_{t-1}(i) exp(-y_i(a_{t-1} f_{t-1}(x_i))) exp(-y_i(a_t f_t(x_i))) \\
&= \frac{1}{Z_t}\frac{1}{Z_{t-1}} D_{t-1}(i) exp(-y_i(a_{t-1} f_{t-1}(x_i) + a_t f_t(x_i))) \\
&= \frac{1}{Z_t Z_{t-1} Z_{t-2}} D_{t-2}(i) exp(-y_i(a_{t-2} f_{t-2}(x_i) + a_{t-1} f_{t-1}(x_i) + a_t f_t(x_i)))
\end{aligned}$$

We can see that there is an emerging pattern:

$$D_{T+1}(i) = \frac{1}{\prod_t Z_t} D_1(i) exp(-y_i(\sum_t a_t f_t(x_i)))$$

Recall from the algorithm initialization $D_1(i) = \frac{1}{m}$, therefore:

$$D_{T+1}(i) = \frac{1}{m}\frac{1}{\prod_t Z_t} exp(-y_i(\sum_t a_t f_t(x_i)))$$

Also we know that $g(x_i) = \sum_t a_t f_t(x_i)$:

$$D_{T+1}(i) = \frac{1}{m}\frac{1}{\prod_t Z_t} exp(-y_i g(x_i))$$

This gives us the right hand side and concludes the proof. $\square$

(ii)

Proof. Knowing that:

$$D_{T+1}(i) = \frac{1}{m}\frac{1}{\prod_t Z_t} exp(-y_i g(x_i))$$

And because the $D_t(i)$ are normalized, all of their sums should be equal to 1, in other words, $\sum_i D_{T+1}(i) = 1$, using this fact we can get the value of $\prod_t Z_t$:

$$\begin{aligned}\sum_i D_{T+1}(i) &= \frac{1}{m}\frac{1}{\prod_t Z_t}\sum_i exp(-y_i g(x_i)) \\ 1 &= \frac{1}{m}\frac{1}{\prod_t Z_t}\sum_i exp(-y_i g(x_i)) \\ \prod_t Z_t &= \frac{1}{m}\sum_i exp(-y_i g(x_i))\end{aligned}$$

The right hand side is the exponential loss of g(x). Now by using the given fact that the one zero loss is less than the exponential loss:

$$\begin{aligned}err(g) &= \frac{1}{m}\sum_i \mathbf{1}[y_i \neq sign(g(x_i))] \\ &\leq \frac{1}{m}\sum_i exp(-y_i g(x_i)) = \prod_t Z_t\end{aligned}$$

This proves that the error of the aggregate classifier is upper bounded by the the product of Z_t. □

(iii)

Proof. Knowing that:

$$Z_t = \sum_i D_t(i) exp(-a_t y_i f_t(x_i)) \tag{8}$$

$$\varepsilon_t = \sum_i D_t(i) 1[y_i \neq f_t(i)] \tag{9}$$

$$1 - \varepsilon_t = \sum_i D_t(i) 1[y_i = f_t(i)] \tag{10}$$

We can get the value of Z_t:

$$\begin{aligned}Z_t &= \sum_i D_t(i) exp(-a_t y_i f_t(x_i)) \\ &= \sum_{y_i \neq f_t(x_i)} D_t(i) exp(a_t) + \sum_{y_i = f_t(x_i)} D_t(i) exp(-a_t) \\ &= \varepsilon_t exp(a_t) + (1-\varepsilon_t) exp(-a_t)\end{aligned}$$

We substitute with the value of $a_t = \frac{1}{2} ln(\frac{1-\varepsilon_t}{\varepsilon_t})$

$$\begin{aligned}
Z_t &= \varepsilon_t exp(\frac{1}{2} ln(\frac{1-\varepsilon_t}{\varepsilon_t})) + (1-\varepsilon_t) exp(-\frac{1}{2} ln(\frac{1-\varepsilon_t}{\varepsilon_t})) \\
&= \varepsilon_t \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} + (1-\varepsilon_t)\sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}} \\
&= \sqrt{\varepsilon_t^2 \frac{1-\varepsilon_t}{\varepsilon_t}} + \sqrt{(1-\varepsilon_t)^2 \frac{\varepsilon_t}{1-\varepsilon_t}} \\
&= \sqrt{\varepsilon_t(1-\varepsilon_t)} + \sqrt{(1-\varepsilon_t)\varepsilon_t} \\
&= 2\sqrt{\varepsilon_t(1-\varepsilon_t)}
\end{aligned}$$

This gives us the right hand side and concludes the proof. □

(iv)

Proof. We are given that each iteration t, $\varepsilon_t = \frac{1}{2} - \gamma_t$ Starting with the left hand side:

$$\begin{aligned}
\prod_t 2\sqrt{\varepsilon_t(1-\varepsilon_t)} &= \prod_t 2\sqrt{(\frac{1}{2}-\gamma_t)(1-(\frac{1}{2}-\gamma_t))} \\
&= \prod_t 2\sqrt{(\frac{1}{2}-\gamma_t)(\frac{1}{2}+\gamma_t))} \\
&= \prod_t 2\sqrt{\frac{1}{4}-\gamma_t^2} \\
&= \prod_t \sqrt{4(\frac{1}{4}-\gamma_t^2)} \\
&= \prod_t \sqrt{1-4\gamma_t^2}
\end{aligned}$$

Which gives us the left hand side of the inequality, to get the right hand side:

$$\begin{aligned}
\prod_t 2\sqrt{\varepsilon_t(1-\varepsilon_t)} &= \prod_t \sqrt{1-4\gamma_t^2} \\
&= \prod_t (1-4\gamma_t^2)^{\frac{1}{2}} \\
&\leq \prod_t (exp(-4\gamma_t^2))^{\frac{1}{2}} \\
&= \prod_t exp(-2\gamma_t^2) \\
&= exp(-2\sum_t \gamma_t^2)
\end{aligned}$$

This gives us the right hand side of the inequality and concludes the proof. □

Problem 3

(i)

With the matrix A, binary vector x_i being hashed to b could be expressed as following:

$$\begin{aligned} b &= Ax_i \\ \begin{bmatrix} b^1 \\ b^2 \\ \cdots \\ b^p \end{bmatrix} &= \begin{bmatrix} A^1 \\ A^2 \\ \cdots \\ A^p \end{bmatrix} x_i \\ &= \begin{bmatrix} (\sum\limits_{l=1}^{n} A^{1l} x_i^l) \bmod 2 \\ (\sum\limits_{l=1}^{n} A^{2l} x_i^l) \bmod 2 \\ \cdots \\ (\sum\limits_{l=1}^{n} A^{pl} x_i^l) \bmod 2 \end{bmatrix} \end{aligned} \tag{11}$$

Take the t-th element of b as an example: $b^t = ((\sum\limits_{l=1}^{n} A^{tl} x_i^l) \bmod 2)$. Since $x_i \in \{0,1\}^n$, supposing there are n_i nonzero elements in x_i, then $\sum\limits_{l=1}^{n} A^{tl} x_i^l$ is actually to randomly pick nonzero elements from the n_i nonzero elements in x_i. And $b^t = ((\sum\limits_{l=1}^{n} A^{tl} x_i^l) \bmod 2)$ is in fact describing whether we pick even or odd number of nonzero elements from all the nonzero elements in x_i. Since A^t is generated uniformly at random, the probability of the t-th entry of b, $b^t = 0$ is $\text{Prob}(b^t = 0) = 1/2$ and that of $b^t = 1$ is also $\text{Prob}(b^t = 1) = 1/2$. And this is true for any t between 1 and p. Because different entries of matrix A are independent with each other, $\text{Prob}(x_i \to b)$ $= \prod\limits_{t=1}^{p} \text{Prob}(A^t x_i \to b^t) = \prod\limits_{t=1}^{p} 1/2 = 1/2^p$.

(ii)

We solve this problem using two methods.

METHOD 1: From part (i), the probability of x_j hashing to any b is $1/2^p$. So the probability of x_j hashing to the particular b that x_i is hashed to is $1/2^p$. In other words, the probability of x_i and x_j hashing to the same vector is $1/2^p$.

METHOD 2: That x_i and x_j hashing to the same vector means $Ax_i = Ax_j$:

$$\begin{bmatrix} (\sum_{l=1}^{n} A^{1l}x_i^l) \bmod 2 \\ (\sum_{l=1}^{n} A^{2l}x_i^l) \bmod 2 \\ \cdots \\ (\sum_{l=1}^{n} A^{pl}x_i^l) \bmod 2 \end{bmatrix} = \begin{bmatrix} (\sum_{l=1}^{n} A^{1l}x_j^l) \bmod 2 \\ (\sum_{l=1}^{n} A^{2l}x_j^l) \bmod 2 \\ \cdots \\ (\sum_{l=1}^{n} A^{pl}x_j^l) \bmod 2 \end{bmatrix} \tag{12}$$

Then

$$\begin{bmatrix} (\sum_{l=1}^{n} A^{1l}[x_i^l - x_j^l]) \bmod 2 \\ (\sum_{l=1}^{n} A^{2l}[x_i^l - x_j^l]) \bmod 2 \\ \cdots \\ (\sum_{l=1}^{n} A^{pl}[x_i^l - x_j^l]) \bmod 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix} \tag{13}$$

which means $A(x_i - x_j) = \mathbf{0}$.

Supposing x_i and x_j have m different elements. Then any $A^t(x_i - x_j) = 0$ for $(1 \leq t \leq p)$ means to pick even number of elements from the m elements. This is the probability of a random variable X with binomial distribution Binomial(m, p_x) being even[1]. Because each entry of matrix A are generated uniformly at random, the probability of any element in A being 0 is equal to that being 1. That means, for any element in the m elements, the probability of it being picked is equal to that of it not being picked. So $p_x = 1/2$. So:

$$\begin{aligned} \text{Prob}(A^t(x_i - x_j) = 0) &= \text{Prob } (\text{X}_{m,p_x} \text{ is even}) \\ &= \frac{1}{2}(1 + (1 - 2p_x)^m) \\ &= \frac{1}{2}(1 + (1 - 2 \times \frac{1}{2})^m) \\ &= \frac{1}{2} \end{aligned} \tag{14}$$

Because different elements of b are independent with each other, Prob$(A(x_i - x_j) = 0)$ $= \prod_{t=1}^{p} \text{Prob}(A^t(x_j - x_j) = 0) = \prod_{t=1}^{p} 1/2 = 1/2^p$.

[1]For reference, check https://math.stackexchange.com/questions/1149270/probability-that-a-random-variable-is-even and https://math.stackexchange.com/questions/2541864/hashing-the-cube-binary-matrix-combinatorics

(iii)

The probability of no collisions among the x_i could be represented as following:

$$\begin{aligned}
\text{Prob (no collisions)} &= 1 - \text{Prob (exist collisions)} \\
&\geq 1 - \sum_{1 \leq i < j \leq m} \text{Prob}(x_i, x_j \text{ collide}) \\
&= 1 - \sum_{1 \leq i < j \leq m} 1/2^p \\
&= 1 - \binom{m}{2} \frac{1}{2^p} \\
&= 1 - \frac{m(m-1)}{2} \frac{1}{2^p} \\
&\geq 1 - \frac{m^2}{2} \frac{1}{2^p}
\end{aligned} \tag{15}$$

If $p \geq 2 \log_2 m$,

$$\begin{aligned}
\text{Prob (no collisions)} &\geq 1 - \frac{m^2}{2} \frac{1}{2^p} \\
&\geq 1 - \frac{m^2}{2} \frac{1}{m^2} \\
&= 1 - 1/2 \\
&= 1/2
\end{aligned} \tag{16}$$

So if $p \geq 2 \log_2 m$, there are no collisions among the x_i with probability at least 1/2.

Problem 4

iii)

Our final regressor is a neural network that has the following architecture:

1. A fully connected input layer with relu activation of 90 units.
2. Dropout layer with 20% drop out ratio.
3. A hidden fully connected layer with relu activation of 64 units.
4. Dropout layer with 20% drop out ratio.
5. A hidden fully connected layer with relu activation of 32 units.
6. Dropout layer with 20% drop out ratio.
7. Batch normalization layer.
8. Output layer with tanh activation layer bounded by the maximum and minimum year interval of 1 unit.

We ran this model with 50 epochs, batch size of 32 and a 10% validation. And we got 5.4469 training loss and 5.5862 validation loss. For the implementation of this neural network we used Keras[1] with a Tensorflow[2] backend.

The only pre processing we tried is features normalization using Scikit learn[3] StandardScalar, and it improved the accuracy of the neural network.

We reached this model after trying several times with different other models and architectures:

1. The first approach we tried was elastic net with 5 fold cross validation on the penalty regularization hyperparameters, this approach gave us a 6.8 training loss and 6.7 validation loss.
2. We tried ridge regression with 5 fold cross validation on the regularization hyperparameters which also gave us similar results nothing less than 6.6 .
3. And then we normalized the features using Scikit learn StandardScalar which didn't improve the results at all, for both models.
4. And Then we decided to give neural networks a try, with a very simple model with an input layer, 1 hidden relu layer, and a linear activation. which gave us a result around 6 for the validation loss.
5. We kept adding layers: dropout layers to prevent over fitting and a bounded output activation to avoid weird predictions (example, year: 2702?).

References

1. Francois Chollet et al.Keras.https://keras.io. 2015.

2. Martin Abadi et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015. url: https://www.tensorflow.org/.

3. F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In:Journal ofMachine Learning Research12 (2011), pp. 2825–2830.

4. H. Zou and T. Hastie. "Regularization and Variable Selection via the Elastic Net".In:Journal of the Royal Statistical Society: Series B (Statistical Methodology)67.2(2003), pp. 301–320.