CS771A : Assignment 1

Group Name:
E-bot

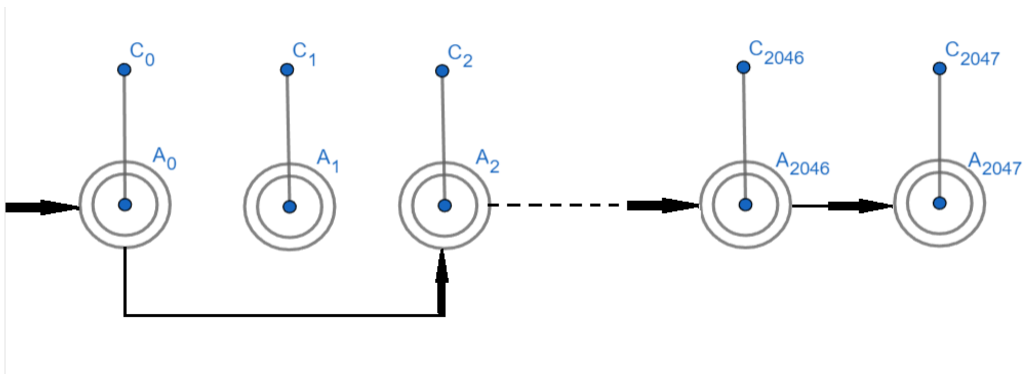
Name	Roll Number
Aditya Kushwaha	200051
Aman Pal	200102
Amit Kumar	200109
Amit Madhesiya	200111
Kaushal Kishor Shukla	200497
Sameer Khan	200853

Submitted to:
Prof Purshottam Kar

1 Task- 1

We have been provided with data from a Sparse CDU PUF with D = 2048 and S = 512 i.e. only 512 of the 2048 CDUs are actually active and the rest 1536 are disconnected and play no role in response generation. However, it is not known which 512 CDUs are active.

Now, the first task is to mathematically design a single sparse linear model which is fit for the given sparse CDU PUF. The following figure shows the string of the CDUs :-



$t_0, t_1, t_2, ..., t_{2047}$ are the output delays at first, second, third,.....,2048th(last) CDU.
$c_0, c_1, c_2, ..., c_{2047}$ are select bits(0/1) or challenges.
$p_0, p_1, p_2, ..., p_{2047}$ are the delays in microseconds if the select bit is 1.

Output delay at 2nd CDU (t_1) depends on t_0, c_1 and p_1, where t_0 is the previous delay, c_1 represents whether there will be a delay in the ith CDU or not (if CDU is connected). p_1 represents the delay in microseconds in the current CDU i.e. 1st CDU if CDU is connected and select bit c_1 is 1. The delay in output for i units is represented by t_{i-1}. We can write :

$$t_1 = \begin{cases} w_1 \cdot c_1 + t_0, & \text{if CDU is connected,} \\ \mathrm{t}_0, & \text{if CDU is disconnected.} \end{cases}$$

So we can define a term w_1

as $w_1 = \begin{cases} p_1, & \text{if CDU is connected,} \\ 0, & \text{if CDU is disconnected.} \end{cases}$

And write the equation as -

$t_1 = w_1 \cdot c_1 + t_0$

It is clear that a similar relation holds for any stage.

$t_i = w_i \cdot c_i + t_{i-1}$

$$w_i = \begin{cases} p_i, & \text{if CDU is connected,} \\ 0, & \text{if CDU is disconnected.} \end{cases}$$

Here, $c_i \in \{0,1\}$ and $w_i \in \mathbb{R}$

t_{-1} can be safely assumed to be zero.

$t_0 = w_0 \cdot c_0$

$t_1 = w_1 \cdot c_1 + t_0$

$\vdots$

$t_{2046} = w_{2046} \cdot c_{2046} + t_{2045}$

$t_{2047} = w_{2047} \cdot c_{2047} + t_{2046}$

Combining,

$t_{2047} = w_{2047} \cdot c_{2047} + w_{2046} \cdot c_{2046} + \ldots + w_1 \cdot c_1 + w_0 \cdot c_0$

$t_{2047} = w^T \cdot C$

Response y is total delay(in microseconds) incurred in the entire chain i.e.

$y = t_{2047}$

$y = w^T \cdot C$

where $C \in \{0,1\}^D$, $w \in \mathbb{R}^D$

This way, a sparse CDU PUF can be broken by a single sparse linear model.

By definition, w has at-most S non-zero coordinates because only S units are connected.

Expression $y = w^T \cdot \phi(c)$ gives the correct response. where $\phi(c) = c$ and c is D-bit 0/1 valued challenge vector. It is also clearly visible that our linear model does not have any bias term (not even a hidden one)

2 Task- 2

Submitted the code.

3 Task- 3

Our Experience with various linear methods

1. Naïve linear regression approach:
   - R-squared value was very poor (approximately -0.23).
   - Discovered negative values in the learned coefficients.
   - Implemented a function to set negative values to zero after each coefficient update.
   - Diverged for step lengths greater than 0.1.
   - Converged with poor performance at step lengths around 0.000001, taking minutes to converge.
   - Concluded that this problem couldn't be solved using the naïve technique due to the need for S-sparse solutions.
2. Projected Gradient Descent(PGD) :
   - Noticed negative values in the coefficients.
   - Replaced negative values with zeros before and applied the high threshold function.
   - Initially, training time was 60 to 70 seconds due to nested loops.
   - Discovered that training time could be significantly reduced by using **vectorization** techniques.
   - Implemented vectorization, reducing the training time to a few seconds.
3. LASSO Technique:
   - Modified the code to incorporate LASSO regularization (L1 regularization) in the loss function and updated gradient term accordingly.
   - LASSO regularization aims to learn a sparse linear model.
   - Tested different values of lambda (regularization hyperparameter) such as 0.1, 1, 2, 3, 4, and 5.

- Discovered that lambda values equal to and less than 1 yielded satisfactory results.
- Encountered negative coefficients despite achieving S-sparse model through LASSO regularization.
- Mean squared error came out large for this method because of negative coefficients.
- Employed the Non-negative LASSO technique to address the issue.

4. Non-negative LASSO Technique:

   - Non-negative LASSO is a variation of LASSO where the coefficients are constrained to be non-negative.
   - In LASSO regularization, the penalty term is the L1 norm of the coefficient vector, promoting sparsity by driving some coefficients to zero.
   - Non-negative LASSO modifies the LASSO approach by including only positive coefficients in the regularization term.
   - By enforcing non-negativity, the Non-negative LASSO technique ensures that all coefficients contribute positively to the model.

Table 1: Summary of Model Performance

Method	Tuned Step Length	Tuned Reg. Parameter	Time Taken (s)	model_err	mae_err
PGD	0.014	NA	4.86	218.61	44.17
LASSO	0.01	0.1	1.41	186.71	517.08
Non-negative LASSO	0.01	0.1	3.86	219.45	50.77

Note :- All performance matric are calculated using 70% -30% splitting(70% training data, 30% testing data) of the given 1600 CRPs

Why did we like Projected Gradient Descent over LASSO technique or Non-negative LASSO technique ?

1. **Sparsity Control:** Lasso uses an L1 regularization term to enforce sparsity, whereas PGD allows for more exact sparsity control. As we know the sparsity of the model beforehand. In PGD, we can explicitly set the desired sparsity level by specifying the number of non-zero coefficients.

2. **Constraints:** PGD allows for the incorporation of constraints on the model coefficients. Here, we have a constraint that coefficients (delay of the PUFs) can not be negative. We can easily incorporate this constraint directly into the optimization process of the PGD.

3. **Performance:** PGD can often provide faster convergence compared to Lasso, especially when dealing with large-scale datasets or high-dimensional feature spaces. The projection step in PGD, which brings the coefficients back to the feasible region, can help avoid unnecessary computations and improve convergence speed.

Additionally, Comparing performance of the above models, we chose the Projected Gradient Descent method over the LASSO technique or Non-negative LASSO technique. Although the Projected Gradient Descent method is taking a little more time, but it is performing best in terms of mean squared error.

4 Task- 4

Hyperparameter tuning for PGD method

We submitted the Projected Gradient Descent method, which has one hyperparameter (step-length) to be tuned. We used grid search technique for tuning. We looked at model error and mean absolute error to choose best value of step length.

Firstly, we used following values of the step-length to know the order of the best performing step length

Table 2: Tuning of step length

Alpha	model_err	mae_err
0.0001	208.78	52.63
0.001	210.03	51.50
0.01	219.33	46.95
0.1	2357.44	24909.30
1	23811.89	267380.56

We observed that the optimal value of the step length is order of 10^{-2}. To know the precise value of step length we calculated errors for the following values

Table 3: Precise tuning of step length

Alpha	model_err	mae_err
0.011	219.83	46.61
0.012	218.61	44.17
0.013	219.31	42.50
0.014	220.00	41.89
0.015	220.53	41.40
0.016	375.68	2278.65
0.017	395.04	2548.06
0.018	420.56	2817.47
0.019	442.22	3086.89

Finally, we concluded that the optimal value of step length is around 0.014