CS771 Assignment 1

Team PAC
Group No.23

Chetanya Bhan 210283
Nikhil Meena 210667
Paturi Bhavya 210712
Somya Gupta 211049

1 Mathematical Derivation of a Linear Model for Sparse CDU PUF

1.1 Mapping Challenge Vectors to Feature Vectors

Let $\phi : \{0,1\}^D \to \mathbb{R}^D$ be a mapping that maps D-bit $0/1$-valued challenge vectors to D-dimensional feature vectors. The feature vector $\phi(c)$ for a given challenge vector c is defined as follows:

- Each coordinate of the feature vector corresponds to a different challenge bit, i.e., $\phi(c) = [\phi_0(c_0), \phi_1(c_1), \ldots, \phi_{D-1}(c_{D-1})] = [p_0, p_1, \ldots, p_{D-1}]$.
- For each coordinate i, $\phi_i(c)$ takes a real value that represents the delay of a particular cell. If the challenge bit is 0, there is no delay, but if the challenge bit is 1, the delay added is p_i seconds from that cell.

1.2 Sparse CDU PUF

Let's assume that we have a sparse CDU PUF, which means that only S out of the D components or cells in the PUF contribute to the final response. In other words, for any given challenge vector c, there exist S indices $i_1, i_2, \ldots, i_S$ such that $\phi_{i_1}(c), \phi_{i_2}(c), \ldots, \phi_{i_S}(c)$ are the only non-zero components of $\phi(c)$.

1.3 Sparse Linear Model

We want to find a sparse linear model $w \in \mathbb{R}^D$ that can mimic the behavior of the CDU PUF for all challenge vectors. This means that for any challenge vector c, the dot product $w^T\phi(c)$ should give the correct response.

We need to construct a sparse linear model w that satisfies the following conditions:

- $\|w\|_0 \leq S$: The norm $\|w\|_0$ represents the number of non-zero coordinates in w. We want to ensure that w has at most S non-zero coordinates.
- $w^T\phi(c)$ gives the correct response for all challenges c.

1.4 Constructing the Sparse Linear Model

To construct such a sparse linear model, we can set the non-zero coordinates of w to be the same indices that correspond to the non-zero components of $\phi(c)$ for any challenge c.

Let $w = [w_0, w_1, \ldots, w_{D-1}]$ be the sparse linear model. For the desired conditions, we set:

- For each index i such that $\phi_i(c_i)$ is non-zero, we set $w_i = 1$.
- For all other indices j, we set $w_j = 0$.

Since the sparse CDU PUF has at most S non-zero components, the constructed linear model w will also have at most S non-zero coordinates.

1.5 Mathematical Model

Consider p_i to be the delay in transmitting the signal in the (i+1)$^{\text{th}}$ CDU given a select bit of 1. Let c_i be the select bit or 'challenge' given to the the (i+1)$^{\text{th}}$ CDU. Let α_i = 1 correspond to an active (i+1)$^{\text{th}}$ CDU and α_i correspond to an inactive (i+1)$^{\text{th}}$ CDU. As we know already it is an S-Sparse CDU PUF, S number of α_i will hold value 1 for S many unknown values of i. Let the total signal delay after passing through the (i+1)$^{\text{th}}$ CDU be given by Δ_i. Then

$$\begin{aligned}
\Delta_0 &= \alpha_0 p_0 c_0 \\
\Delta_i &= \Delta_{\text{i-1}} + \alpha_i p_i c_i \\
\text{Overall delay in signal, } \Delta_{\text{D-1}} &= \alpha_0 p_0 c_0 + \alpha_1 p_1 c_1 + ... + \alpha_{\text{D-1}} p_{\text{D-1}} c_{\text{D-1}} \\
\Delta_{\text{D-1}} &= \sum_{i=0}^{D-1} \alpha_i p_i c_i \\
\Delta_{\text{D-1}} &= \sum_{i=0}^{D-1} (\alpha_i p_i) c_i \\
\Delta_{\text{D-1}} &= (\boldsymbol{\alpha} \cdot \mathbf{p})^T \mathbf{c} \\
\Delta_{\text{D-1}} &= \mathbf{w}^T \mathbf{c} \\
\Delta_{\text{D-1}} &= \mathbf{w}^T \phi(\mathbf{c}) \qquad \text{where, } \mathbf{w} = \boldsymbol{\alpha} \cdot \mathbf{p}, \quad \text{and } \phi(\mathbf{c}) = \mathbf{c}.
\end{aligned}$$

1.6 Correctness of the Linear Model

Now, let's consider the dot product $w^T \phi(c)$ for any challenge vector c.

$$w^T \phi(c) = [w_0, w_1, \ldots, w_{D-1}]^T \cdot [\phi_0(c_0), \phi_1(c_1), \ldots, \phi_{D-1}(c_{D-1})] = w_0\phi_0(c_0) + w_1\phi_1(c_1) + \ldots + w_{D-1}\phi_{D-1}(c_{D-1})$$

Since we constructed w such that only the non-zero components of $\phi(c)$ have non-zero coordinates in w, all other terms in the dot product become zero. Thus, we are left with:

$$w^T \phi(c) = w_i \phi_i(c_i) \quad \text{(where } i \text{ represents the non-zero indices of } \phi(c))$$

Since $\phi_i(c_i)$ represents the correct response of the corresponding component for the given challenge c, the dot product $w^T \phi(c)$ gives the correct response.

Therefore, by constructing a sparse linear model w as described above, we can break a sparse CDU PUF, where $\|w\|_0 \leq S$, and the dot product $w^T \phi(c)$ gives the correct response for any challenge c.

2 Code

See ***submit.py*** file for code submission.

3 Methods Used

3.1 Coordinate Descent

Coordinate descent is an optimization algorithm that successively minimizes along coordinate directions to find the minimum of a function. At each iteration, the algorithm determines a coordinate or coordinate block, then minimizes over the corresponding coordinate hyperplane while fixing all other coordinates or coordinate blocks.

Given:

$$f() = 1/2||Y - Xw||_2^2$$

for min f()
selecting any coordinate j

$$\frac{\partial f()}{\partial w_j} = 0$$

$$\frac{\partial f()}{\partial w_j} = x_j^T(Xw - Y)$$

$$\frac{\partial f()}{\partial w_j} = x_j^T(X_j w_j + \Sigma_{i\neq j} X_i w_i - Y)$$

therefore,

$$x_j^T(X_j w_j + \Sigma_{i\neq j} X_i w_i - Y) = 0$$

$$w_j = \frac{X_j^T(y - \Sigma_{i\neq j} X_i w_i)}{X_j^T X_j}$$

LASSO regularization using soft threshold

l:regularization parameter

$$W[j] = (sign(w[j])([|w[j]| - l]_+)$$

where,

$$[t]_+ = \max\{t, 0\}$$

This shrinks the magnitude of the parameter coordinate update toward zero.

Feature selection

$$indices = \{i : W[i] > 0\}$$

$$W'[indices] = argmin_w[1/2\Sigma_{m=1}^n(\Sigma_{j\in indices} W_j^T X_j^m - Y^m)^2]$$

$$W'(j \neq indices) = 0$$

Hard threshold

def HT(z,k)
we retain top k coordinates of z and set others to zero.
k=s,(s=512)
z=W'

3.2 Projected Gradient Descent

The Projected Gradient Descent (PGD) method is an optimization algorithm commonly used in machine learning for solving constrained optimization problems. It is particularly useful when the optimization problem involves a constraint set, and we want to find the solution that satisfies the constraints.

Given an optimization problem of the form:

$$\min_{\mathbf{w}} f(\mathbf{w})$$

subject to the constraint $\mathbf{w} \in C$, where $f(\mathbf{w})$ is the objective function and C is the constraint set.

The PGD algorithm iteratively updates the solution $\mathbf{w}$ by taking small steps in the direction of the negative gradient of the objective function and projecting the updated solution onto the constraint set C.

The steps of the PGD algorithm can be summarized as follows:

1. Initialize the solution $\mathbf{w}_0$.
2. For $t = 1, 2, \ldots, T$, where T is the number of iterations:

- Compute the gradient $\nabla f(\mathbf{w}_{t-1})$ of the objective function with respect to $\mathbf{w}$ at the current solution $\mathbf{w}_{t-1}$.
- Update the solution by taking a step in the negative gradient direction:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \nabla f(\mathbf{w}_{t-1}),$$

  where η is the step size or learning rate.
- Project the updated solution $\mathbf{w}_t$ onto the constraint set C:

$$\mathbf{w}_t = \arg\min_{\mathbf{w} \in C} \|\mathbf{w} - \mathbf{w}_t\|_2^2,$$

  where $\|\cdot\|_2$ represents the Euclidean norm.

3. Return the final solution $\mathbf{w}_T$.

The projection step ensures that the updated solution remains within the feasible region defined by the constraint set C. This is crucial for solving constrained optimization problems.

The step size η plays an important role in the convergence and performance of the algorithm. It needs to be carefully chosen to balance the trade-off between convergence speed and stability.

The PGD algorithm provides a flexible and effective approach for optimizing constrained machine learning problems by iteratively updating the solution in the direction of steepest descent and projecting onto the constraint set. It has been widely applied in various domains, including support vector machines, logistic regression, and neural networks.

3.3 FISTA

The Fast Iterative Shrinkage Thresholding Algorithm, is an improvised version of the Proximal Gradient Descent method of Iterative Shrinkage Thresholding Algorithm (ISTA) used to solve convex optimization problems with regularization terms leading to sparse solutions. It is faster than ISTA because of added acceleration to the prox function, which makes it reach a convergence speed much faster than the traditional gradient descent methods. The FISTA algorithm for a LASSO regression is

- Initialise x_1 to some arbitrary vector and set z_1 to x_1.
- Create variables like μ_0 (initialise), $\mu_m = \frac{1+\sqrt{1+4\mu^2{}_{\text{m-1}}}}{2}$, $\gamma_m = \frac{1-\mu_m}{\mu_m+1}$
- Update z_{m+1} = prox(x_m-$\frac{2}{\beta}$$A^T$($Ax_m$-y)), where prox(z)=u, u_i=sign(z_i)(|z_i|-α)$_+$
- Update x_{m+1}=(1-γ_m).z_{m+1} + $\gamma_m z_m$

We initialised x_1 and z_1 to the least squares solution as it was the most convenient initialisation to compute and also produced good results. We set another variable, the Lipschitz constant (L) to the squared norm of the input matrix. Small values of L lead to faster convergences, so this value was chosen. The FISTA algorithm function takes in a matrix of challenges, the vector of responses, a variable t for updating ..., a variable l used in soft-thresholding (shrinkage), a variable maxiter that decides the maximum number of iterations and a variable n that decides number of nested iterations carried out per iteration. In every iteration, we update z and x variables according to the FISTA algorithm. After each round of nested iterations, we sparsify x, separate the non-zero indices and find the least squares solution for data corresponding to these particular non-zero coordinates only and set this to x for the next iteration. Soon, the solution converges.

4 Hyperparameter Tuning

4.1 Coordinate Descent

The only hyperparameter used in this algorithm was l. By plotting the graphs of abs(l1) we found the optimum value of l1. Further after the negative values were discarded, we changed the value of l2 similarly. And after taking only highest 512 values and solving for them we similarly changed the value of l3 by seeing the variations in the values of abs(c).

4.2 Projected Gradient Descent

The hyperparameters used in this algorithm are eta (step size) and iter (number of iterations). We test for the following values:

- eta_values=[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500]
- iter_values=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 40, 50, 60, 100]

We record metrics like Train Time, Model Error, MAE Error (Mean Absolute Error) and Support Recovery for every combination. We assess performance of the parameters using MAE error. We keep track of the parameters giving least MAE error and consider them the 'best parameters'.

4.3 FISTA

There were four parameters to be optimised for the best performance- l, t, n, maxiter. We carried this out using a grid search. We iterated through possible values of all these variables. The following values were tested

- l_values = [0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5, 10, 50],
- t_values = [0.1, 0.5, 1.0, 5, 10, 50, 100, 500, 1000],
- n_values = [0.005, 0.01, 0.1, 0.5, 1.0,3, 5, 10, 50],
- maxit_values = [10,11,9,8,7,6,5,4,12,13,14].

We iterated through majority of each possible combination of these four variables which were taking time less than 15 seconds, trained and tested on the split datasets, recording metrics like time elapsed and mean absolute percentage error [between the predicted values and the correct values] for every combination. We consider performances with mean absolute percentage error value $< 10^{-7}$ only and call the one with least elapsed time the best combination. We also keep note of the best parameters in each iteration.

The best parameters after testing and changing them further turned out to be l=0.000001, t=50, n=25, maxit=8. Further we introduced the variable k=6 for tuning the iterations in the internal loop.

5 Conclusion

All 3 algorithms are giving similar mae error but Coordinate descent is much slower. And, Proximal gradient descent one is having high mae errors as the train and test data are separated.

Table 1: Results

FISTA		
Datasets	Train Time	Mean Absolute Error
X_trn, X_tst	8.645	1.945e-07
90:10 X_trn	6.433	0.0017297
80:20 X_trn	5.695	0.794
70:30 X_trn	4.426	20.934

Table 2: Results

Projected Gradient Descent		
Datasets	Train Time	Mean Absolute Error
X_trn, X_tst	9.16059	2.83691e-07
90:10 X_trn	7.73935	0.77373
80:20 X_trn	5.59380	4.88674
70:30 X_trn	5.42205	400.60546

Table 3: Results

Coordinate Descent		
Datasets	Train Time	Mean Absolute Error
X_trn, X_tst	165.55412	2.9345e-07

So, in final submission we used FISTA which gave best results.

6 References

[1] `https://www.youtube.com/watch?v=JRerBpNggNO&ab_channel=ConstantineCaramanis`
[2] `https://en.wikipedia.org/wiki/Coordinate_descent`
[3] `https://gist.github.com/agramfort/ac52a57dc6551138e89b`