771Crackers

Assignment 1

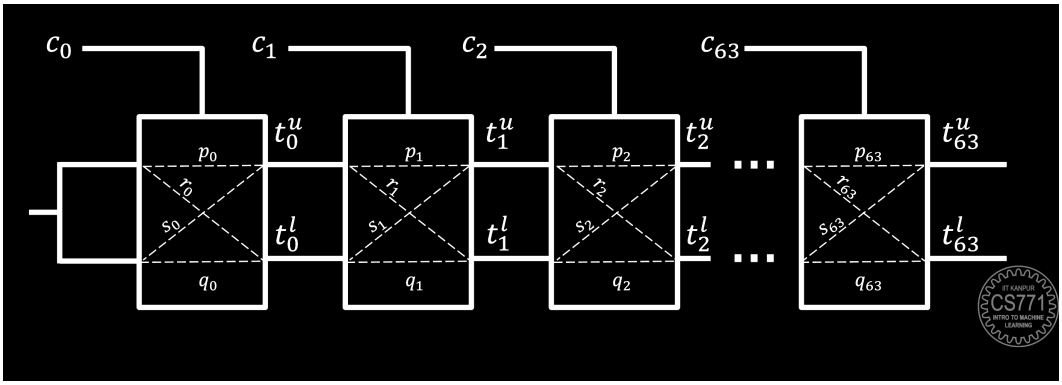
Abstract

In the 1st part of the question, our objective is to demonstrate the vulnerability of the Companion Arbiter PUF (CAR-PUF) to a single linear model, contrary to Melbo's belief. We achieve this by simplifying the CAR-PUF into two linear models and analyzing their behavior. We proceed systematically, starting from basic principles and gradually advancing towards our goal. We first decompose the CAR-PUF into two linear models, denoted as (u, p) and (v, q), which can accurately predict its outputs. By simplifying the CAR-PUF in this manner, we lay the groundwork for our subsequent analysis. Throughout this process, we adhere to the instruction that our mapping function, $\phi(c)$, should depend solely on the challenge vector, c, and universal constants. Next, we proceed step by step, considering necessary transformations and adjustments, to arrive at a linear model with a different dimensionality compared to the individual 32-bit models. Through rigorous mathematical derivation, we define the mapping function $\phi(c)$ and determine the dimensionality of the feature vector, W, required for the linear model. Our approach is guided by the principles outlined in the problem statement, ensuring that our solution remains aligned with the given constraints, objectives and everything learnt during the discussion hours and lectures.

In the 3rd part of the assignment, we evaluate the performance of the sklearn.svm.**LinearSVC** model and the sklearn.linear_model.**LogisticRegression** model on predicting the responses of the Companion ArbiteR PUF (CAR-PUF). We explore the impact of various hyperparameters on training time and test accuracy. Specifically, we vary the loss hyperparameter in LinearSVC (hinge vs squared hinge), along with other hyperparameters such as C, tol, and penalty. We present our findings in terms of training time, test accuracy, and the influence of different hyperparameter settings on the model's performance. Please be aware that the data presented in the tables within the solution for part 3 has been tested on a local machine, and there may be slight variations if run on Google Colab or another platform.

1 Mathematical Derivation of Linear Model for CAR-PUF

Following the approach outlined in our lecture slides, we initially derive the linear model for a single Physical Unclonable Function (PUF) operating with 32-bit challenges. At this stage, our analysis applies to both the working (w) and reference (r) PUFs. We begin by specifying the input to both PUFs as the 32-bit challenges ($c_i's$). Each bit in the 32-bit binary vector (**c**) corresponds to a pair of multiplexers, each with four unknown delays (represented as p_i, q_i, r_i, and s_i) through which the incoming signal traverses.



Here, t_i^u and t_i^l represent the times at which the signal departs from the i_{th} MUX pair. Expressing t_i^u in terms of t_{i-1}^u, t_{i-1}^l and c_i, we obtain:

$$t_i^u = (1 - c_i) \cdot (t_{i-1}^u + p_i) + c_i \cdot (t_{i-1}^l + s_i) - (1)$$

$$t_i^l = (1 - c_i) \cdot (t_{i-1}^l + q_i) + c_i \cdot (t_{i-1}^u + r_i) - (2)$$

Thus following the lecture slides we have, $\Delta_i = t_i^u - t_i^l$. Subtracting equations (1), (2) we get, $\Delta_i = \Delta_{i-1} \cdot d_i + \alpha_i \cdot d_i + \beta_i$, where α_i, β_i depend on constants that are indeterminable from the physical/ measurable perspective and thus we call them system constants and are given by:

$$\alpha_i = (p_i - q_i + r_i - s_i)/2 \quad \beta_i = (p_i - q_i - r_i + s_i)/2$$

where p_i, q_i, r_i, s_i are system parameters. Also d_i is governed by the challenge bits/input $(c_i's)$:

$$d_i = (1 - 2 \cdot c_i)$$

Δ_{-1} = 0. Moreover, observing the recursion unfold carefully we can simplify this relation further:

$$\Delta_0 = \alpha_0 \cdot d_0 + \beta_0$$

$$\Delta_1 = \alpha_0 \cdot d_1 \cdot d_0 + (\alpha_1 + \beta_0) \cdot d_1 + \beta_1$$

$$\Delta_2 = \alpha_0 \cdot d_2 \cdot d_1 \cdot d_0 + (\alpha_1 + \beta_0) \cdot d_2 \cdot d_1 + (\alpha_2 + \beta_1) \cdot d_2 + \beta_2$$

$$\vdots$$

The only Δ we require is Δ_{31}, the last output.

$$\Delta_{31} = w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_{31} \cdot x_{31} + \beta_{31} = \mathbf{w}^T \cdot \mathbf{x} + \mathbf{b} = \mathbf{W}^T \cdot \mathbf{X}$$

where **w, x** are 32 dimensional vectors and **W, X** are 33 dimensional, just including the bias term $W_{32} = \beta_{31}$ and $X_{32} = 1$. Each term of **w, x** being:

$$b = \beta_{31} \qquad w_0 = \alpha_0 \qquad w_i = \alpha_i + \beta_{i-1} \qquad x_i = \prod_{j=i}^{31} d_i$$

$$Response(\gamma) = \frac{1 + sign(w^T \cdot x + b)}{2}$$

Remark: The only parameter we need for predicting the output of Δ_{31}, given a new challenge, is the vector **w**, which we extract out from pre-existing efficient linear models which use the feature vector **x** for multiple challenge response pairs as available data for training.

1.1 Calculating the Required Response

So now, returning back to our original question about analysing and deducing w, x, b for the CAR-PUF problem, let the linear model defined for the working PUF, as a normal PUF analysed above, be with the parameters (u, p) and similarly for the reference PUF be (v, q).

$$\Delta_{31}^{w} = \mathbf{u}^T \cdot \mathbf{x} + \mathbf{p} \qquad \Delta_{31}^{r} = \mathbf{v}^T \cdot \mathbf{x} + \mathbf{q}$$

where **x** is the common feature vector composed of the 32 bits of a challenge and **u, v** are made of system parameters for both the PUF's respectively.

As for the question the response is based on the absolute difference:

$$D = |\Delta_{31}^{w} - \Delta_{31}^{r}| \qquad Response(\gamma) = \begin{cases} 0, & \text{if } D \leq \tau \\ 1, & \text{if } D > \tau \end{cases}$$

we use a simple trick to make the constraint more obvious as it was in a normal linear model for PUF's. We just square the difference D(Δ_{31}^{w} - Δ_{31}^{r}) to get the simplified response and remove the modulus:

$$\gamma = \begin{cases} 0, & \text{if } (\Delta_{31}^{w} - \Delta_{31}^{r})^2 \leq \tau^2 \\ 1, & \text{else} \end{cases} \quad \rightarrow \quad \gamma = \begin{cases} 0, & \text{if } (\Delta_{31}^{w} - \Delta_{31}^{r})^2 - \tau^2 \leq 0 \\ 1, & \text{else} \end{cases}$$

Further modification leads to the final response being very similar to that of a normal PUF linear model response. Thus the final response we receive, applying the sign function, is:

$$\gamma = \frac{1 + sign[(\Delta_{31}^{w} - \Delta_{31}^{r})^2 - \tau^2]}{2}$$

1.2 Simplifying $\Delta_{31}^{w} - \Delta_{31}^{r}$

As we know from earlier discussion:

$$\Delta_{31}^{w} = u^T \cdot x + p = (u_0, u_1, u_2, \cdots, u_{31}, p)^T \cdot (x_0, x_1, x_2, \cdots, x_{31}, 1)$$

$$\Delta_{31}^{r} = v^T \cdot x + q = (v_0, v_1, v_2, \cdots, v_{31}, q)^T \cdot (x_0, x_1, x_2, \cdots, x_{31}, 1)$$

$$\Delta_{31}^{w} - \Delta_{31}^{r} = (u - v)^T \cdot x + (p - q)$$

$$= (u_0 - v_0, u_1 - v_1, u_2 - v_2, \cdots, u_{31} - v_{31}, p - q)^T \cdot (x_0, x_1, x_2, \cdots, x_{31}, 1)$$

So, we can rename the system variables $u_i - v_i$ as z_i's and $p - q$ as y and write this difference as a new 33 dimensional linear model as follows:

$$\Delta_{31}^{w} - \Delta_{31}^{r} = \mathbf{z}^T \cdot \mathbf{x} + \mathbf{y} = (z_0, z_1, z_2, \cdots, z_{31}, y)^T \cdot (x_0, x_1, x_2, \cdots, x_{31}, 1)$$

Now,

$$\gamma = \frac{1 + sign[(\mathbf{Z}^T \cdot \mathbf{X})^2 - \tau^2]}{2} \quad \textbf{where} \quad \mathbf{Z}^T \cdot \mathbf{X} = \mathbf{z}^T \cdot \mathbf{x} + \mathbf{y}, Z_{32} = y, X_{32} = 1$$

where z_i's depend upon the corresponding working and reference PUF system parameters or in laymen terms the internal delays in each MUX used. Also y is a constant bias term that comes from the 2 bias terms used to model the linear model for both the PUF's independently and **x** is the challenge binary vector.

Now the issue we face ahead of us is how to make a linear model even after we are using the square of the linear model made above.

1.3 Feature Transformation and Model Augmentation

Inspired from the binomial expansion:

$$(a_1 + a_2 + \cdots + a_n)^2 = a_1^2 + a_2^2 + \ldots + a_n^2 + 2(a_1 \cdot a_2 + a_1 \cdot a_3 + \ldots + a_{n-1} \cdot a_n)$$

$$(a_1 + a_2 + \cdots + a_n)^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} (a_i \cdot a_j)$$

We can thus change the feature vector **x** to something inspired from the above equation and generate a linear model of different dimensions than the original one inside the squaring bracket in the equation:

$$(Z^T \cdot X)^2$$

Thus the **new dimensionality** of the newly generated feature vector will be:

$$^{33}C_2 = 33 \cdot 16 = \textbf{528 features}$$

In terms of the original features the new feature vector X_{new} contains the following set of features as it's components:

$$S = \{x_i \cdot x_j \mid 0 \leq i < j \leq 32\}$$

As x_{32} = 1 (constant),

$$S = \{x_i \cdot x_j \mid 0 \leq i < j \leq 31\} \cup \{x_i \mid 0 \leq i \leq 31\}$$

So now our new model and it's response is:

$$\gamma = \frac{1 + sign[\mathbf{A}^T \cdot X_{new} - \tau^2]}{2} \rightarrow X_{new}, \textit{ A are } \textbf{\textit{528}} \textit{ dimensional vectors}$$

Also for our model to work efficiently if we incorporate the $-\tau^2$ term into our model, the **total** number of **features** becomes **529** and the features being :

$$S = \{x_i \cdot x_j \mid 0 \leq i < j \leq 32\} \cup \{1\}$$

The specific contents of vector A are inconsequential since they are solely determined by u, v, p, q, and τ and will be acquired through regression of the linear model ultimately. However, for a given set S, I can establish a set C representing the components of vector A as follows:

$$C = \{z_i \cdot z_j \mid 0 \leq i < j \leq 32\} \cup \{-\tau^2\}$$

1.4 Defining the MAP ϕ : $\{0,1\}^{32} \rightarrow \mathbb{R}^k$, Dimension k

The 5 basic steps needed for defining the map we need are :

1. For each $i = 0$ to 31, calculate $d_i = 1 - 2 \cdot c_i$, where $c_i's \in \mathbf{c}$.
2. For each $i = 0$ to 31, calculate $x_i = \prod_{j=i}^{31} d_j$, where $x_i's \in \mathbf{x}$.
3. Append 1 to the end of vector **x**, resulting in a 33-dimensional vector $\mathbf{X}$.
4. For $0 \leq i < j \leq 31$, multiply X_i with X_j, $X_i \in \mathbf{X}$ to generate a $32 \cdot 33/2 = 528$ dimensional vector X_{new} which has the elements of set:
$$S = \{x_i \cdot x_j \mid 0 \leq i < j \leq 31\} \cup \{x_i \mid 0 \leq i \leq 31\}$$
5. Thus we obtain 528-dimensional vector output.

This will successfully generate the required output. Also the value of **k** - The **dimensionality** of the output vector is **528**. [Not including the bias term that the model will generate seperately.]

2 Solution for Part 2:

*Zipped **Solution** to Assignment 1*

3 Perfomance Analysis for Models

(a) Performance Comparison of LinearSVC with Different Loss Functions:

Loss Function	Total Features	Model Train Time (s)	Map Time (s)	Test Accuracy
Hinge	528	9.2509	0.0748	0.9891
Squared Hinge	528	9.7067	0.0705	0.9914

Analysis: From the table, we observe that the model trained with the squared hinge loss achieves a slightly higher test accuracy compared to when trained with the hinge loss. However, the difference in accuracy is insignificant. Interestingly, the model trained with the squared hinge loss takes slightly longer to train compared to the model trained with the hinge loss. This indicates that the squared hinge loss may lead to a more complex optimization problem. Nonetheless, the difference in training time between the two models is relatively small.

Possible reasons for the observed differences:

- The squared hinge loss penalizes outliers more strongly than the hinge loss, potentially leading to improved generalization performance.
- The optimization problem associated with the squared hinge loss may require more iterations to converge, leading to increased training time.
- The dataset characteristics and the specific challenges posed by the CAR-PUF may favor one loss function over the other, influencing the model's performance.

(b) Performance Comparison based on the value of C:

- **Analysis for LinearSVC with different C values:**

C Value	Total Features	Model Train Time (s)	Map Time (s)	Test Accuracy
Low	528	11.8110	0.0779	0.9899
Medium	528	9.8451	0.0765	0.9912
High	528	10.1107	0.0768	0.9899

- **Analysis for LogisticRegression with different C values:**

C Value	Total Features	Model Train Time (s)	Map Time (s)	Test Accuracy
Low	528	1.5308	0.0940	0.9871
Medium	528	1.6730	0.0933	0.9922
High	528	2.7172	0.0999	0.9932

From the analysis, we observe the following:

For LinearSVC:

- The model trained with a medium C value achieves the highest test accuracy among the three settings.
- The model trained with a low C value has the highest training time, which may indicate that a lower C value leads to a more complex optimization problem.
- The model trained with a high C value has similar test accuracy to the low and medium C value models but with a slightly lower training time.

For LogisticRegression:

- Similar to LinearSVC, the model trained with a medium C value achieves the highest test accuracy.
- LogisticRegression generally has significantly lower training times compared to LinearSVC, regardless of the C value.
- The high C value model in LogisticRegression achieves the highest test accuracy, indicating that a higher regularization strength may improve generalization performance.

Overall, the choice of C value significantly affects the training time and test accuracy of both LinearSVC and LogisticRegression models. A medium C value tends to strike a balance between model complexity and generalization performance.

(c) Effect of Variation in Tolerance for LinearSVC and LogisticRegression Models:

For part (c) of the analysis, we evaluate the performance of the LinearSVC and LogisticRegression models with varying values of the `tol` hyperparameter. The `tol` hyperparameter controls the tolerance for the optimization algorithm, with lower values potentially leading to longer training times but potentially improved convergence and accuracy. We present the results in the following table:

Model	Tolerance	Total Features	Model Train Time	Map Time	Test Accuracy
LinearSVC	Low	528	9.902s	0.0770s	0.9906
LinearSVC	Medium	528	9.740s	0.0789s	0.9909
LinearSVC	High	528	1.169s	0.0951s	0.9359
LogisticRegression	Low	528	1.350s	0.1120s	0.9907
LogisticRegression	Medium	528	1.260s	0.1090s	0.9907
LogisticRegression	High	528	1.050s	0.1190s	0.9854

From the table, we observe the following:

- For LinearSVC, varying the tolerance parameter leads to significant differences in training time, with the low tolerance resulting in the longest training time and the high tolerance resulting in the shortest training time.
- However, the test accuracy of the LinearSVC model decreases as the tolerance increases, indicating that higher tolerance values may lead to poorer generalization performance.
- For LogisticRegression, the differences in training time across different tolerance values are less pronounced compared to LinearSVC.
- Similar to LinearSVC, the test accuracy of the LogisticRegression model also shows a decreasing trend as the tolerance increases, albeit to a lesser extent.

These results suggest that the choice of tolerance parameter can have a significant impact on both the training time and the test accuracy of the models. Lower tolerance values may lead to longer training times but potentially better performance, while higher tolerance values may result in faster training times but poorer generalization performance. It is crucial to strike a balance between training time and model performance when selecting the tolerance parameter.

Concluding Part 3:

From the above data analysis, it seems that **LogisticRegression with high C value** performs the best in terms of test accuracy (**0.9932**) and an approximate model train time of (**2s**) and map time (**0.0999s**). This model strikes a **good balance** between **accuracy** and computational **efficiency**.

The Team

Name	Email	Roll number
Prem Kansagra	premk22@iitk.ac.in	220816
Dobariya Jenil Bharatbhai	dobariyajb22@iitk.ac.in	220385
Mridul Gupta	mridulg22@iitk.ac.in	220672
Harshit	harshit22@iitk.ac.in	220436
Priyanshu Singh	spriyanshu22@iitk.ac.in	220830
Ankit Kaushik	kankit22@iitk.ac.in	220158