CS771 | Introduction to Machine Learning
Assignment 1 - Report

Hritik Kumar
210451
hritik21@iitk.ac.in

Rishav Dev
210847
rishavd21@iitk.ac.in

Deepanshu
210311
deepanshud21@iitk.ac.in

Kamal Kishor
210483
kamalk21@iitk.ac.in

Akansha patel
210081
akanshap21@iitk.ac.in

Amisha Patel
210119
amishap21@iitk.ac.in

Team Name: Decoders

Abstract

In this assignment, we demonstrate the vulnerability of Melbo's Companion Arbiter PUF (`CAR-PUF`) to linear models, providing a detailed mathematical derivation of a mapping function that breaks the security of `CAR-PUF`. Subsequently, we implement and analyze the linear model using `sklearn's LinearSVC` and `LogisticRegression`, exploring the impact of hyperparameters on training time and test accuracy to validate the susceptibility of `CAR-PUF` to linear modeling techniques.

1 Companion Arbiter PUF

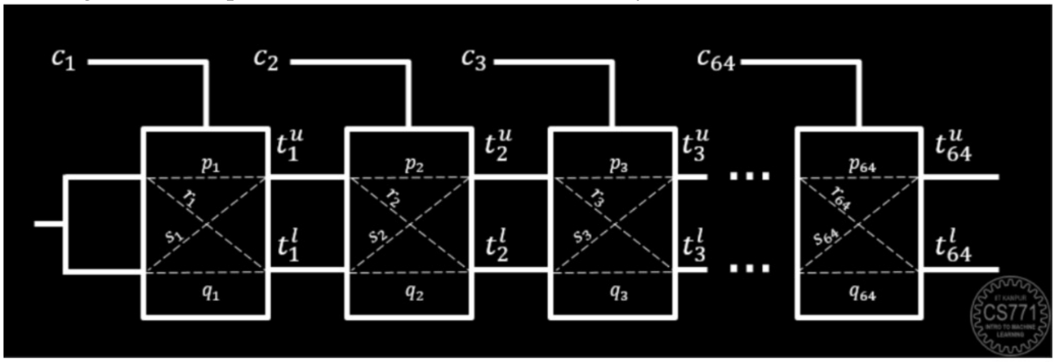
1.1 Arbiter PUFs - A Brief Introduction

Arbiter Physically unclonable Functions are hardware systems that capitalize on unpredictable differences in data transmission speed in different iterations of the same design to implement security.

They consist of a series of multiplexers each having a selection bit. A particular set of selection bits is said to form a "question/challenge", depending on these selection bits, the signal that reaches the end of the PUF first is said to be the answer. The answer to a particular challenge is therefore unique to the hardware of that particular PUF.

1.2 Using ML to find the time taken by upper signal to reach the finish line

It has been found that knowing the time taken by a relatively smaller number of challenges, a model can be trained to predict the time taken by any other challenge for a particular arbiter PUF, the analysis for which is shown below:



t_i^u is the (unknown) time at which the upper signal leaves the i-th PUF. t_i^l is the (unknown) time at which the upper signal leaves the i-th PUF. All the PUFs are different so that p_i, r_i, s_i and q_i are distinct. Therefore, the answer is 0 if $t_{31}^u < t_{31}^i$ and 1 otherwise.

(Here we have assumed that zero-based indexing which implied $t_{-1}^u = t_{-1}^l = 0$).

Now,

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

Now,

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

which turn out to be

$$\Delta_i = t_i^u - t_i^l = (1 - c_i) (p_i - q_i) + c_i(s_i - r_i) + \Delta_{i-1} - 2c_i\ (\Delta_{i-1})$$

$$= (\Delta_{i-1})(1 - 2c_i) + (1\text{-}2c_i) (p_i - q_i + r_i - s_i)/2 + (p_i - q_i - r_i + s_i)/2$$

$$\Delta_{i-1}\ d_i + d_i\ x_i + y_i$$

where ($d_i = (p_i - q_i + r_i - s_i)/2$ and $y_i = (p_i - q_i - r_i + s_i)/2$)

For i=1,

$$t_1^u = p_1 + c_1(s_1 - p_1) + 0 - c_1(0)$$
$$= p_1 + c_1(s_1 - p_1)$$
$$\Delta_i = d_1 x_1 + y_1$$

For i=2,

$$t_2^u = p_2 + c_2(s_2 - p_2) + p_1 + c_1(s_1 - p_1) - c_2(p_1 - q_2 - c_1(p_1 - q_1 + r_1 - c_1))$$

$$= p_2 + p_1 + c_2(s_2 - p_2) + c_1(s_1 - p_1) - c_2(d_1 x_1 + y_1)$$

Let say $z_1 = s_1 - p_1$, $z_2 = s_2 - p_2$ and $z_i = s_i - p_i$,

$$= p_1 + p_2 + c_2 z_2 + c_1 z_1 - c_2 d_1 x_1 - c_2 y_1$$

$$= p_1 + p_2 + c_2(z_2 - y_1) + c_1 z_1 - c_2 d_1 x_1$$

$$= [z_1, (z_2 - y_1 - 2x_1)] \begin{bmatrix} c_1 \\ c_2 \\ c_2c_1 \end{bmatrix}$$

For i=3,

t_3^u = p_3^u + c_3(s_3 - p_3) + p_u + c_2(s_2 - p_2) + p_1 - c_1 (s_1 - p_1) - c_2(p_1 - q_1 - c_1(p_1 - q_1 + r_1 - s_1)) + c_3 (p_2 - q_2 - c_2 (p_2 - q_2 + r_2 - s_2) + (p_1 - q_1 - c_1 (p_1 - q_1 + r_1 - s_1) (1- 2 c_2))

= p_3 + p_2 + p_1 + c_3 (s_3 - p_3) + c_2 (s_2 - p_2) + c_1 (s_1 - p_1) + c_2 (z_2 - y_1) - $c_2d_1x_1$ - c_3 (d_2x_2 + y_2 + $d_2d_1x_1$ + d_2y_1)

= (p_3 + p_2 + p_1) + c_3 (z_3 - y_2) + c_2 (z_2 - y_1) + c_1y_1 - $c_2d_1x_1$ - c_3d_2(x_2 + y_1) - $c_3d_2d_1x_1$

$$= [y_1, (z_2 - y_1), -(x_1), (z_3 - y_2), -(x_2 + y_1)] \begin{bmatrix} c_1 \\ c_2 \\ c_2d_1 + c_3d_2d_1 \\ c_3 \\ c_3d_2 \end{bmatrix}$$

Similarly, for t_3^u ,

$$t_{32}^u = \sum_{j=0}^{32} p_j \; +$$

$[y_1, (z_2 - y_1), -(x_1), (z_3 - y_2), (-(x_2 + y_1)), (z_3 - y_2), (-(x_2 + y_1)), (z_4 - y_3), (-(x_3 + y_2)) \ldots$

continuing from the equation below ,

$$(z_{32} - y_{31}), (-(x_{31} + y_{30}))] \begin{bmatrix} c_1 \\ c_2 \\ c_{32}d_{31}\ldots d_1 + c_{31}d_{30}\ldots d_1 + c_{30}d_{29}\ldots d_1 + \ldots + c_2d_1 \\ c_3 \\ c_{32}d_{31}\ldots d_2 + c_{31}d_{30}\ldots d_2 + c_3d_2 \\ \vdots \\ c_{32} \\ c_{32}d_{31} \end{bmatrix}$$

After Substituting d_i = 1 - 2c_i ϕ can be further simplied as :

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{32} \\ c_{32}c_{31}\cdots c_2 \\ c_{32}c_{31}\cdots c_3 \\ c_{32}c_{31}\cdots c_4 \\ \vdots \\ c_{32}c_{31} \end{bmatrix}$$

$$t_{32}^u = \mathrm{w}^T\phi + b$$

Clearly, we can see that dimension is 62x1.

Now question 3 ,

$$\text{if } (t^l_{32})^1 \text{ - } (t^l_{32})^0 > 0 \text{ ; } r^0 = 1$$

$$\text{where } r^0 = \text{response } 0$$

$$(t^l_{32})^1 \text{ - } (t^l_{32})^0 < 0 \text{ ; } r^0 = 0$$

Thus ,

$$r^0 = [1 + \text{sign } ((t^l_{32})^1 \text{ - } (t^l_{32})^0)]/2$$

$$(t^l_{32})^1 \text{ - } (t^l_{32})^0 = [(w^l)^1 \text{ - } (w^l)^0]^T \phi_c + [(b^l)^1 \text{ - } (b^l)^0]$$

$$= (\widetilde{\mathbf{w}})^T \phi_c + \widetilde{b}$$

ϕ_c remains the same
We can clearly see that the dimension remain the same that means 62X1.

2 Tasks

2.1 Task 1 : Mathematical Derivation

From previous parts, we can conclude now that the dimension for $\widetilde{w}$ is 62 as also obtained from the code as explained.

2.2 Task 2 : Results Of Code

The final results of the code are as follows:

Table 1: Results

D	t_train	t_map	Accuracy y0	Accuracy y1
62.0	0.8458	0.0814	0.9863	0.9962

The python code for `my_fit()` function we used is as follows:

```
def my_fit( X_train, y0_train, y1_train ):
################################
#  Non Editable Region Ending  #
################################
    X_mapped = my_map(X_train)
    mod0 = linear_model.LogisticRegression(C=10, max_iter=2100)
    mod1 = linear_model.LogisticRegression(C=10, max_iter=2100)
    mod0.fit(X_mapped, y0_train)
    mod1.fit(X_mapped, y1_train)
    w0 = mod0.coef_
    b0 = mod0.intercept_
    w1 = mod1.coef_
    b1 = mod1.intercept_
    return w0, b0, w1, b1
```

The python code for `my_map()` function we used is as follows:

```
def my_map(X):
################################
#  Non Editable Region Ending  #
################################
        n_sam, n_col = X.shape
```

```python
        ans = np.copy(X)
        prod_X = np.ones(n_sam)

        for i in range(n_col - 1, 0, -1):
                prod_X *= X[:, i]
                if i is not n_col-1:
                        ans = np.column_stack((ans, prod_X))
        return ans
```

2.3 Task 3 : Varying Hyperparameters

Report outcomes of the following experiments with both the `sklearn.svm.LinearSVC` and `sklearn.linear_model.LogisticRegression` methods when used to learn the linear model. In particular, report how various `hyperparameters` affected training time and test accuracy using tables and/or charts with both `LinearSVC` and `LogisticRegression` methods.

2.3.1 a.) Changing the loss hyperparameter in LinearSVC (hinge vs squared hinge)

The loss function assesses the model's performance on a dataset. sklearn.svm.LinearSVC supports two types of loss functions: hinge and squared hinge.

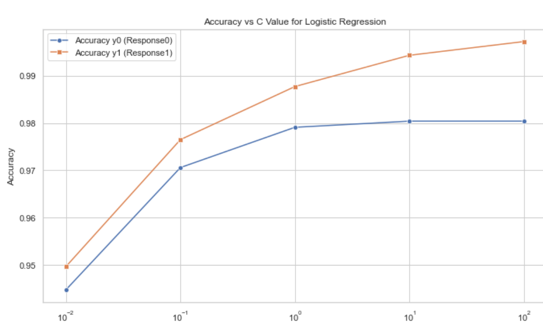
Table 2: Changing the `loss` hyperparameter in LinearSVC

Loss Hyper-parameter	Training time (in sec)	accuracy y0 (in %)	accuracy y1 (in %)
Hinge	0.8458 s	0.9863	0.9962
Squared Hinge	2.263s	0.9862	0.9962

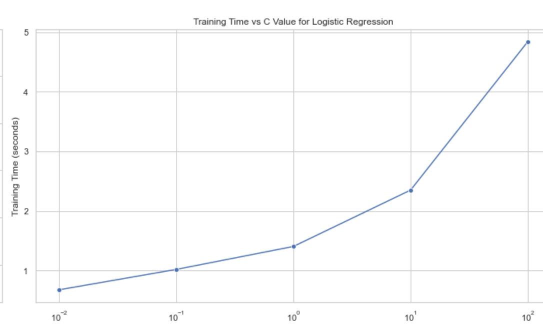
2.3.2 Setting C in LinearSVC and LogisticRegression to high/low/medium values.

Adjusting the regularization parameter C in LinearSVC and LogisticRegression significantly impacts model performance. High C values prioritize accurate training data classification, potentially overfitting, while low values encourage generalization but may reduce training accuracy. Medium C values balance model complexity and generalization for optimal performance.

For LogisticRegression:



(a) Test Accuracy Comparison



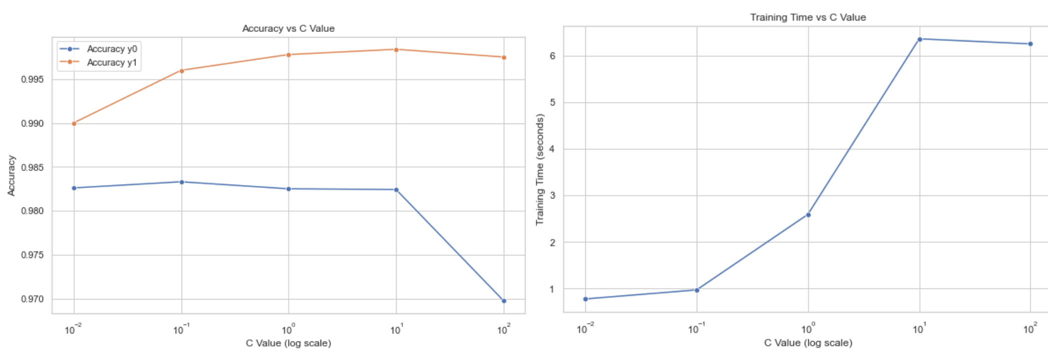
(b) Training Time Comparison

Figure 1: Changing the `C` hyperparameter in LogisticRegression

Table 3: Changing the `C` hyperparameter in LogisticRegression

C Hyper-parameter	Training time (in sec)	Accuracy y0 (in %)	Accuracy y1 (in %)
0.01	0.687	0.9448	0.9497
0.1	1.082	0.9705	0.9765
1	1.2498	0.9791	0.9877
10	2.314	0.9804	0.9943
100	4.2822	0.9804	0.9972

For LinearSVC:



(a) Test Accuracy Comparison (b) Training Time Comparison

Figure 2: Changing the `C` hyperparameter in LinearSVC

Table 4: Changing the `C` hyperparameter in LinearSVC

C Hyper-parameter	Training time (in sec)	accuracy y0 (in %)	accuracy y1 (in %)
0.01	0.6718	0.9826	0.9900
0.1	0.8242	0.9833	0.9960
1	2.2474	0.9825	0.9978
10	5.7440	0.9824	0.9984
100	8.8217	0.9658	0.9982

2.3.3 Changing the penalty (regularization) hyperparameter in LinearSVC and LogisticRegression (l2 vs l1)

For LinearSVC:

Table 5: Changing the `penalty` hyperparameter in LinearSVC

Penalty Hyperparameter	Training time (in sec)	Accuracy y0 (in %)	Accuracy y1 (in %)
l1	14.3080s	0.9825	0.9976
l2	1.4278s	0.9825	0.9978

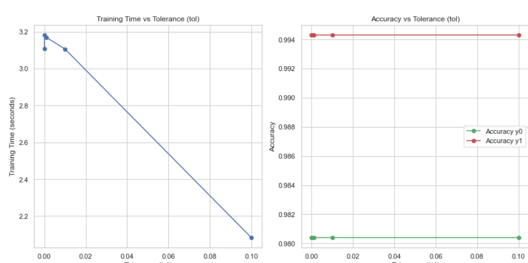
For LogisticRegression:

Table 6: Changing the `penalty` hyperparameter in LogisticRegression

Penalty Hyperparameter	Training time (in sec)	Accuracy y0 (in %)	Accuracy y1 (in %)
l1	42.200	0.9804	0.9976
l2	1.4417	0.9804	0.9940

2.3.4 Changing tol in LinearSVC and LogisticRegression to high/low/medium values.

For LogisticRegression:

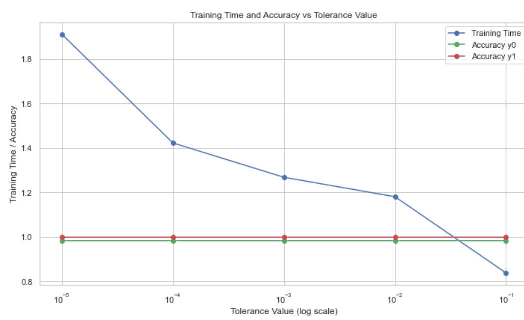


(a) Test Accuracy And Training Time Comparison

Table 7: Changing the `tol` hyperparameter in LogisticRegression

tol Hyper-parameter	Training time (in sec)	Accuracy y0 (in %)	Accuracy y1 (in %)
1e-1	2.0836321	0.9804	0.9943
1e-2	3.0153645	0.9804	0.9943
1e-3	3.0141805	0.9908	0.9944
1e-4	3.0465015	0.9908	0.9945
1e-5	2.92900929	0.9908	0.9943

For LinearSVC:



(a) Test Accuracy And Training Time Comparison

Table 8: Changing the `tol` hyperparameter in LinearSVC

tol Hyper-parameter	Training time (in sec)	Accuracy y0 (in %)	Accuracy y1 (in %)
1e-1	0.8819	0.9827	0.9975
1e-2	1.0154	0.9825	0.99778
1e-3	1.2182	0.9825	0.9978
1e-4	1.5169	0.9825	0.9978
1e-5	1.7629	0.9825	0.9978