Derivations and Analysis for Physically Unclonable Function Models

Aditya Suman
220081
Electrical Engineering
IIT Kanpur
saditya22@iitk.ac.in

Akansha Ratnakar
220093
Chemical Engineering
IIT Kanpur
akanshar22@iitk.ac.in

Priyanshu Gujraniya
220825
Civil Engineering
IIT Kanpur
priyanshug22@iitk.ac.in

Aman Kumar Sriwastav
220116
Material Science and Engineering
IIT Kanpur
amanks22@iitk.ac.in

Ashutosh Anand
220239
Economics
IIT Kanpur
aashutosh22@iitk.ac.in

Abstract

This paper presents our approach to the first assignment of the CS771 course. We develop a linear model for Arbiter PUFs by examining how challenge bits affect signal delays. Our analysis covers both a 16-dimensional linear feature space and a 256-dimensional quadratic feature space, leading us to recommend a degree-2 polynomial kernel SVM for effective classification. Furthermore, we propose a method based on Non-negative Least Squares to estimate individual stage delays. We also report the performance of our models and analyze how hyperparameter tuning impacts training time and prediction accuracy.

1 ML-PUF Linear Model Derivation

Arbiter Physically Unclonable Functions are hardware systems that capitalise on unpredictable differences in data transmission speed in different iterations of the same design to implement security. They consist of a series of multiplexers (mux'es, hereafter) each having a selection bit. A particular set of selection bits is said to form a "challenge", and 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.

Preprint. Under review.

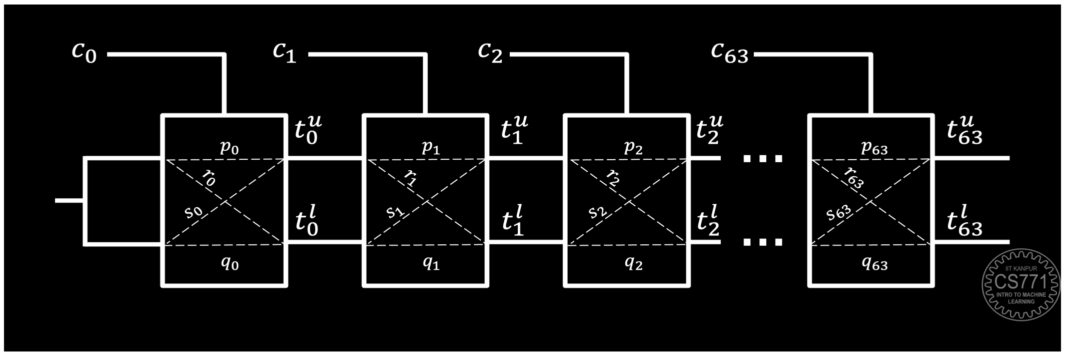

Figure 1: A simple Arbiter PUF with 64 stages

Let t_i^u be the time at which the upper signal leaves the i-th PUF
Let t_i^l be the time for the lower signal leaves the i-th PUF
All PUFs are different so that stage delays $p_i, q_i r_i, s_i$.
The PUF response for an N-stage PUF (indexed 0 to N-1, e.g., N=8 implies stages 0 to 7) is determined by comparing the final arrival times, e.g., response is 0 if $t_{N-1}^l < t_{N-1}^u$ and 1 otherwise.
(We assume zero-based indexing and $t_{-1}^u = t_{-1}^l = 0$).

The stage transition times are given by:

$$t_n^u = (1 - c_n)(t_{n-1}^u + p_n) + c_n(t_{n-1}^l + s_n) \tag{1}$$
$$t_n^l = (1 - c_n)(t_{n-1}^l + q_n) + c_n(t_{n-1}^u + r_n) \tag{2}$$

where $c_n \in \{0, 1\}$ is the challenge bit for stage n.

Adding equations (1) and (2):

$$t_n^u + t_n^l = t_{n-1}^u + t_{n-1}^l + (1 - c_n)(p_n + q_n) + c_n(r_n + s_n)$$

By induction :

$$t_7^u + t_7^l = \sum_{i=0}^{7} [(1 - c_i)(p_i + q_i) + c_i(r_i + s_i)] \tag{3}$$

From the lecture slides, the difference $\Delta_i = t_i^u - t_i^l$ follows:

$$\Delta_i = d_i \Delta_{i-1} + \alpha_i d_i + \beta_i$$

where

$$d_i = (1 - 2c_i), \quad \alpha_i = \frac{p_i - q_i + r_i - s_i}{2}, \quad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

Given the initial condition $\Delta_{-1} = 0$, we get:

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

Let us define:

$$A_i = \alpha_i d_i + \beta_i$$

Then the first few values of Δ_i are:

$$\Delta_0 = A_0$$
$$\Delta_1 = d_1 A_0 + A_1$$
$$\Delta_2 = d_2 d_1 A_0 + d_2 A_1 + A_2$$

The general solution for $N = 8$ stages (i.e., $i = 0$ to 7) is:

$$\Delta_7 = \sum_{k=0}^{7} A_k \prod_{j=k+1}^{7} d_j \tag{4}$$

Substituting A_k:

$$t_7^u - t_7^l = \sum_{i=0}^{7} [\alpha_i(1-2c_i)+\beta_i] \prod_{j=i+1}^{7} (1-2c_j) = \sum_{i=0}^{7} [(p_i-q_i)(1-c_i)+c_i(s_i-r_i)] \prod_{j=i+1}^{7} (1-2c_j) \tag{5}$$

Adding (3) and (5) and dividing by 2 gives t_7^u:

$$t_7^u = \frac{1}{2}\sum_{i=0}^{7}[(1-c_i)(p_i+q_i)+c_i(r_i+s_i)] + \frac{1}{2}\sum_{i=0}^{7}[(1-c_i)(p_i-q_i)+c_i(s_i-r_i)] \prod_{j=i+1}^{7}(1-2c_j) \tag{6}$$

Define the alternative parameters as:

$$\begin{aligned}
\alpha_i' &= \frac{p_i + q_i + r_i + s_i}{4} \\
\beta_i' &= \frac{p_i + q_i - r_i - s_i}{4} \\
\gamma_i' &= \frac{p_i - q_i + s_i - r_i}{4} \\
\delta_i' &= \frac{p_i - q_i + r_i - s_i}{4}
\end{aligned}$$

Equation (6) can be simplified to (using $d_i = 1 - 2c_i$):

$$t_7^u = \sum_{i=0}^{7} \alpha_i' + \sum_{i=0}^{7} \beta_i' d_i + \sum_{i=0}^{7} \gamma_i' \prod_{j=i+1}^{7} d_j + \sum_{i=0}^{7} \delta_i' \prod_{j=i}^{7} d_j \tag{7}$$

Define product terms for $N = 7$:

- $P_i = \prod_{j=i}^{7} d_j = d_i d_{i+1} ... d_7$ for $i = 0, ..., 7$. ($P_7 = d_7$)
- $P_i' = \prod_{j=i+1}^{7} d_j = d_{i+1} d_{i+2} ... d_7$ for $i = 0, ..., 7$. ($P_7' = 1$, $P_i' = P_{i+1}$ for $i = 0, ..., 6$)

The expression (7) becomes:

$$\begin{aligned}
t_7^u &= \sum_{i=0}^{7} \alpha_i' + \sum_{i=0}^{7} \beta_i' d_i + \sum_{i=0}^{7} \gamma_i' P_i' + \sum_{i=0}^{7} \delta_i' P_i \\
&= \left(\sum_{i=0}^{7} \alpha_i' + \gamma_7' P_7'\right) + \sum_{i=0}^{7} \beta_i' d_i + \sum_{i=0}^{6} \gamma_i' P_i' + \sum_{i=0}^{7} \delta_i' P_i \\
&= \left(\sum_{i=0}^{7} \alpha_i' + \gamma_7'\right) + \sum_{i=0}^{7} \beta_i' d_i + \sum_{i=0}^{6} \gamma_i' P_{i+1} + \sum_{i=0}^{7} \delta_i' P_i
\end{aligned}$$

This can be written in matrix form $t^u(c) = \mathbf{W}^T \boldsymbol{\phi}(c) + b$, where $b = \sum_{i=0}^{7} \alpha_i' + \gamma_7'$. Absorbing the bias b into the weight vector using a constant feature $\phi_1' = 1$, we get:

$$t^u(c) = \mathbf{W}'^T \boldsymbol{\phi}'(c)$$

The structure of $\mathbf{W}'$ and $\boldsymbol{\phi}'(c)$ depends on the exact feature map derived from (7).

$$\mathbf{W}' = \begin{pmatrix} b \\ \beta'_0 \\ \beta'_1 \\ \vdots \\ \beta'_6 \\ \delta'_0 \\ \delta'_1 + \gamma'_0 \\ \delta'_2 + \gamma'_1 \\ \vdots \\ \delta'_6 + \gamma'_5 \\ \delta'_7 + \gamma'_6 + \beta'_7(?) \end{pmatrix} \quad \text{and} \quad \boldsymbol{\phi}'(c) = \begin{pmatrix} 1 \\ d_0 \\ \vdots \\ d_6 \\ P_0 = d_0 d_1 \dots d_7 \\ P_1 = d_1 d_2 \dots d_7 \\ \vdots \\ P_7 = d_7 \end{pmatrix}$$

Let's assume the linear model $t^u(c) = \mathbf{W}'^T \boldsymbol{\phi}'(c)$ uses a feature vector $\boldsymbol{\phi}'(c) \in \mathbb{R}^{16}$. Differences between delays of two PUFs (PUF1 and PUF0):

$$\delta_l(c) = t_1^l(c) - t_0^l(c) = (\mathbf{W}'^T_{1l} - \mathbf{W}'^T_{0l})\boldsymbol{\phi}'(c) = \mathbf{W}'^T_{\delta l}\boldsymbol{\phi}'(c) \tag{8}$$
$$\delta_u(c) = t_1^u(c) - t_0^u(c) = (\mathbf{W}'^T_{1u} - \mathbf{W}'^T_{0u})\boldsymbol{\phi}'(c) = \mathbf{W}'^T_{\delta u}\boldsymbol{\phi}'(c) \tag{9}$$

where,

$$\mathbf{W}'_{\delta l}, \mathbf{W}'_{\delta u} \in \mathbb{R}^{16}.$$

The output model using the transformed (quadratic) feature space $\tilde{\boldsymbol{\varphi}}(c)$:

$$\begin{aligned} \tilde{\mathbf{W}}^T \tilde{\boldsymbol{\varphi}}(c) + \tilde{b} &= -(\delta_l(c) \cdot \delta_u(c)) \\ &= -(\mathbf{W}'^T_{\delta l}\boldsymbol{\phi}'(c)) \cdot (\mathbf{W}'^T_{\delta u}\boldsymbol{\phi}'(c)) \quad \text{(Using } \boldsymbol{\phi}' \text{ based on context)} \\ &= -\left(\sum_{i=1}^{16} w'_{\delta li}\phi'_i(c)\right)\left(\sum_{j=1}^{16} w'_{\delta uj}\phi'_j(c)\right) \\ &= \sum_{i=1}^{16}\sum_{j=1}^{16}(-w'_{\delta li}w'_{\delta uj})(\phi'_i(c)\phi'_j(c)) \end{aligned} \tag{*}$$

Since bias was absorbed into $\mathbf{W}'$, the bias term for this quadratic model is $\tilde{b} = 0$.

Final Linear Model (in Quadratic Space): The feature map is $\tilde{\varphi}(c) = (\varphi'_i(c)\varphi'_j(c))_{1 \le i,j \le 16} \in \mathbb{R}^{256}$, where $\varphi'_i(c)$ are components of $\boldsymbol{\phi}'(c)$. The linear model weights are $\tilde{W} \in \mathbb{R}^{256}$ with components $\tilde{W}_{ij} = -w'_{\delta li}w'_{\delta uj}$. The bias is $\tilde{b} = 0$. The prediction for challenge c (assuming $c \in \{0,1\}^8$ as stated is:

$$r(c) = \frac{1 + \text{sign}(\tilde{\mathbf{W}}^T \tilde{\boldsymbol{\varphi}}(c))}{2}$$

---

2 Dimensionality of the Feature Space

From the previous section, the base feature vector $\boldsymbol{\varphi}'(c)$ has dimensionality $D = 16$. The expanded feature vector $\tilde{\varphi}(c)$ contains all pairwise products of these base features:

$$\tilde{\varphi}(c) = (\varphi'_i(c)\varphi'_j(c))_{1 \le i,j \le 16} \in \mathbb{R}^{256}$$

The dimensionality of this expanded feature vector is $\tilde{D} = D^2 = 16^2 = 256$.

$$\tilde{\boldsymbol{\varphi}}(\boldsymbol{c}) = \boldsymbol{\varphi}'(\boldsymbol{c})^T \boldsymbol{\varphi}'(\boldsymbol{c}) = \begin{pmatrix} 1 & d_0 & \dots & d_6 & d_0 d_1 \dots d_7 & d_1 d_2 \dots d_7 & \dots & d_7 \end{pmatrix} \begin{pmatrix} 1 \\ d_0 \\ \vdots \\ d_6 \\ d_0 d_1 \dots d_7 \\ d_1 d_2 \dots d_7 \\ \vdots \\ d_7 \end{pmatrix}$$

---

3 Kernel SVM for ML-PUF Classification

We investigate using a kernel SVM to classify responses for an 8-bit binary challenge $c \in \{0,1\}^8$. The goal is to find a suitable kernel.

3.1 Polynomial Feature Space

Consider a feature map $\Phi(c)$ including linear terms and unique pairwise products:

$$\Phi(c) = (c_1, \dots, c_8, c_1 c_2, c_1 c_3, \dots, c_7 c_8)$$

This map contains 8 linear terms and $\binom{8}{2} = 28$ pairwise products (since $c_i^2 = c_i$ for binary c_i). The total dimensionality is $8 + 28 = 36$.

3.2 The Kernel Trick

A kernel function $K(c, c')$ computes the inner product $\langle \Phi(c), \Phi(c') \rangle$ in the 36-dimensional space without explicit mapping. If the data is linearly separable in this space, the kernel SVM can classify perfectly.

3.3 Kernel Derivation for $\Phi(c)$

The inner product is:

$$K(c, c') = \langle \Phi(c), \Phi(c') \rangle = \sum_{i=1}^{8} c_i c_i' + \sum_{1 \le i < j \le 8} (c_i c_j)(c_i' c_j')$$

Using $\langle c, c' \rangle^2 = \langle c, c' \rangle + 2 \sum_{1 \le i < j \le 8} (c_i c_i')(c_j c_j')$, we isolate the sum:

$$\sum_{1 \le i < j \le 8} (c_i c_j)(c_i' c_j') = \frac{1}{2}[\langle c, c' \rangle^2 - \langle c, c' \rangle]$$

Substituting back into $K(c, c')$:

$$K(c, c') = \langle c, c' \rangle + \frac{1}{2}[\langle c, c' \rangle^2 - \langle c, c' \rangle] = \frac{1}{2}\langle c, c' \rangle^2 + \frac{1}{2}\langle c, c' \rangle$$

3.4 Comparison to a Standard Polynomial Kernel

The standard polynomial kernel of degree $d = 2$ is $K_{\text{poly}}(x, y) = (\gamma \langle x, y \rangle + r)^2$.

$$K_{\text{poly}}(c, c') = \gamma^2 \langle c, c' \rangle^2 + 2\gamma r \langle c, c' \rangle + r^2$$

Matching $K(c, c')$ requires $\gamma^2 = 1/2 \implies \gamma = 1/\sqrt{2}$ and $2\gamma r = 1/2 \implies r = 1/(2\sqrt{2})$. With these parameters, $K_{\text{poly}}(c, c') = \frac{1}{2}\langle c, c' \rangle^2 + \frac{1}{2}\langle c, c' \rangle + \frac{1}{8}$. This matches $K(c, c')$ up to the constant $1/8$, which does not affect the SVM decision boundary.

3.5 Conclusion

- **Recommended Kernel Type:** Polynomial with $d = 2$, $\gamma \approx 0.707$ and $r \approx 0.354$.
- **Justification:** This kernel corresponds to the inner product in the feature space $\Phi(c)$. It captures linear and pairwise interactions between challenge bits.

---

4 Delay Recovery by Inverting an Arbiter PUF Model

We aim to recover 256 non-negative stage delays $x_{\text{delays}} \in \mathbb{R}^{256}_{\geq 0}$ that yield a given 65-dimensional Arbiter PUF linear model $y = (w_0, w_1, \ldots, w_{63}, b)^T \in \mathbb{R}^{65}$.

4.1 Step 1: Forward Model

We construct the sparse linear system $Ax_{\text{delays}} = y$, where $A \in \mathbb{R}^{65\times 256}$. $x_{\text{delays}} = (p_0, q_0, r_0, s_0, \ldots, p_{63}, q_{63}, r_{63}, s_{63})^T$. The model is defined as follows (for $N = 64$ stages, 0-63):

1. **Intermediate delays:**
   For $i = 0, \ldots, 63$, define:
   $$\alpha_i = \frac{1}{2}(p_i - q_i + r_i - s_i), \quad \beta_i = \frac{1}{2}(p_i - q_i - r_i + s_i)$$
2. **Model parameters:**
   $$w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1} \quad \text{for } 1 \leq i \leq 63, \quad b = \beta_{63}$$
3. **Coefficient matrix A:**
   The linear system $Ax_{\text{delays}} = y$ encodes the above relationships. The matrix $A \in \mathbb{R}^{65\times 256}$ is sparse, with at most 8 non-zero entries per row. Columns are grouped into 64 blocks of 4 (corresponding to p_i, q_i, r_i, s_i). The structure is:
   $$A = \begin{pmatrix} A^{(0)} & 0 & 0 & \cdots & 0 \\ B^{(1)} & A^{(1)} & 0 & \cdots & 0 \\ 0 & B^{(2)} & A^{(2)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & B^{(63)} & A^{(63)} \\ 0 & \cdots & 0 & 0 & B^{(64)} \end{pmatrix}$$
   where the non-zero blocks are:
   - **Row 0** ($w_0 = \alpha_0$):
     $$A^{(0)} = \left[\tfrac{1}{2}, -\tfrac{1}{2}, \tfrac{1}{2}, -\tfrac{1}{2}\right] \quad \text{(block 0)}$$
   - **Rows** $r = 1$ **to** 63 ($w_r = \alpha_r + \beta_{r-1}$):
     $$B^{(r)} = \left[\tfrac{1}{2}, -\tfrac{1}{2}, -\tfrac{1}{2}, \tfrac{1}{2}\right] \quad (\text{block } r-1), \quad A^{(r)} = \left[\tfrac{1}{2}, -\tfrac{1}{2}, \tfrac{1}{2}, -\tfrac{1}{2}\right] \quad (\text{block } r)$$
   - **Row 64** ($b = \beta_{63}$):
     $$B^{(64)} = \left[\tfrac{1}{2}, -\tfrac{1}{2}, -\tfrac{1}{2}, \tfrac{1}{2}\right] \quad \text{(block 63)}$$
4. **Sparsity summary:**
   Rows 0 and 64 contain 4 non-zero entries each. Rows 1 to 63 contain 8 non-zero entries each.

4.2 Step 2: Inversion under Non-Negativity

To recover a physically valid solution ($x_{\text{delays}} \geq 0$), we solve the Non-negative Least Squares (NNLS) problem:

$$\min_{x \in \mathbb{R}^{256}} \|Ax - y\|_2^2 \quad \text{subject to } x \geq 0.$$

This can be solved using custom sparse solvers like projected gradient or coordinate descent, adapted for the non-negativity constraint $x \geq 0$. This yields a non-negative delay vector x_{delays} that best reproduces the model y.

---

5 Code to solve the ML-PUF problem

Code Submitted

---

6 Code to solve the Arbiter PUF inversion problem

Code Submitted

---

7 Outcomes of Experiments

7.1 Training Set

The model was trained on the public `trn.txt` and tested on the public `tst.txt` using different sets of hyperparameters. The accuracy and training time for each configuration were recorded.

(a) Effect of Changing the `loss` Hyperparameter in LinearSVC

The loss function defines how well the model fits the data. The `sklearn.svm.LinearSVC` implementation provides two options for the loss function: `hinge` and `squared_hinge`.

The observations reported below were obtained using the following hyperparameters: $C = 1.0$, `tol` = $1e{-}4$, `max_iter` = 2000, and `dual` = `True`.

Table 1: Results of LinearSVC experiments over 5 trials

Loss Function	Average Training Time (s)	Average Test Accuracy
`hinge`	0.2622	0.9706
`squared_hinge`	0.1651	0.9537

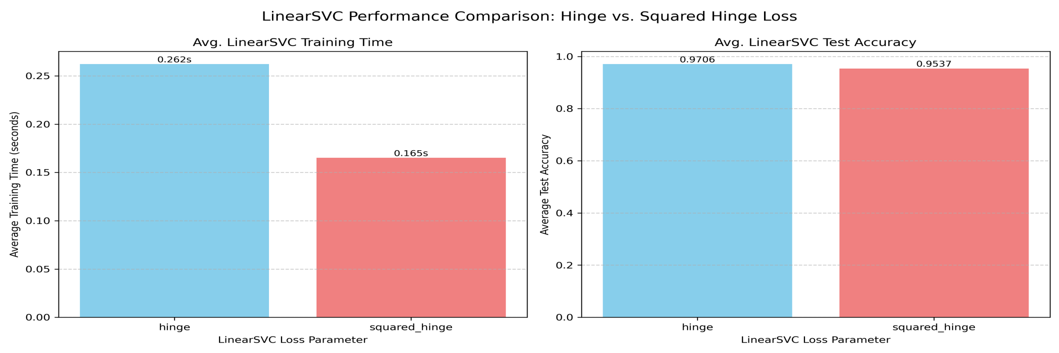

Figure 2: Loss Hyperparameters

(b)Effect of changing the C hyperparameter in LinearSVC and Logistic Regression model

Regularization is a technique used to prevent overfitting by penalizing large coefficients in the model. The C parameter represents the inverse of regularization strength, where smaller values of C correspond to stronger regularization. In both Logistic Regression and Linear SVC, adjusting C influences how closely the model fits the training data. Increasing C tightens the fit, potentially leading to overfitting, while decreasing C encourages simpler decision boundaries, helping generalization but risking underfitting.

We made the observations that are mentioned below using the hyperparameters loss = Squared Hinge, tolerance = 1e-4, penalty = L2 and dual = False for LinearSVC and tolerance = 1e-4, penalty = L2 and dual = False for Logistic Regression:

Table 2: Effect of varying C on LinearSVC and LogisticRegression (5 trials each)

Classifier	C	Avg Training Time (s)	Avg Test Accuracy
LinearSVC	0.01	0.0681	0.9069
	1.0	0.1053	0.9537
	100.0	0.3547	0.9581
LogisticRegression	0.01	0.0190	0.8806
	1.0	0.0302	0.9587
	100.0	0.0521	0.9750

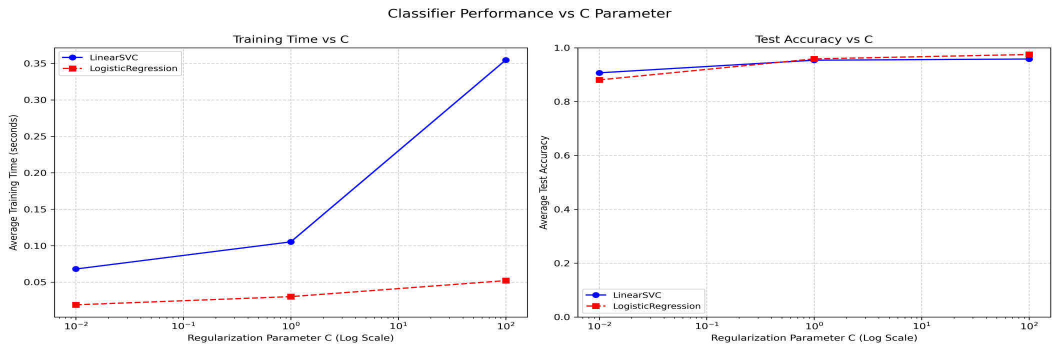

Figure 3: Classifier Performance with C parameter.

(c) Effect of changing the tol hyperparameter in LinearSVC and Logistic Regression model

The tol hyperparameter, representing tolerance, sets the convergence threshold for optimization algorithms in LinearSVC and LogisticRegression. It determines when the iterative optimization process stops by measuring the change in the objective function or coefficients between iterations. A smaller tol implies stricter convergence criteria, prolonging training but potentially improving accuracy. Conversely, a larger tol speeds up training but might compromise precision. Balancing computational efficiency and model accuracy hinges on selecting an appropriate tol value. We made the observations that are mentioned below using the hyperparameters loss = Squared Hinge, C=1, penalty = L2 and dual = False for LinearSVC and C=1, penalty = L2, solver = 'lbfgs' and dual = False for Logistic Regression:

Table 3: Effect of varying tolerance parameter on LinearSVC and LogisticRegression (5 trials each)

Classifier	Tolerance	Avg Training Time (s)	Avg Test Accuracy
LinearSVC	1.0×10^{-2}	0.0739	0.9537
	1.0×10^{-4}	0.0823	0.9537
	1.0×10^{-6}	0.1112	0.9537
LogisticRegression	1.0×10^{-2}	0.0250	0.9450
	1.0×10^{-4}	0.0324	0.9587
	1.0×10^{-6}	0.0652	0.9587

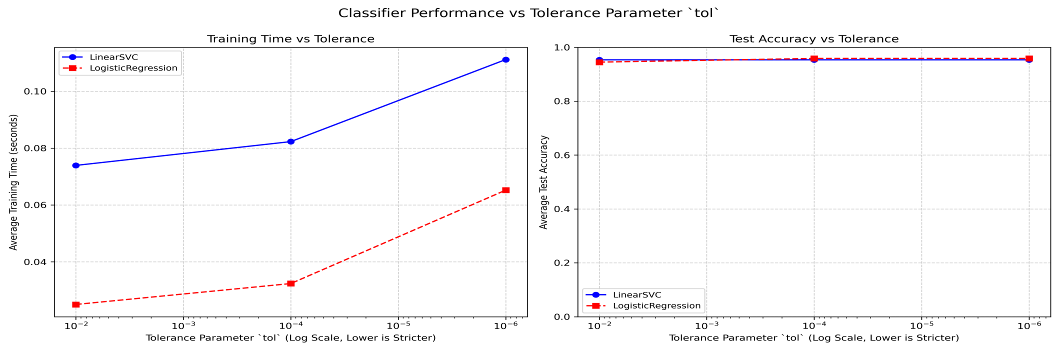

Figure 4: Classifier Performance vs Tolerance Parameter 'tol'.

(d) Effect of changing the penalty hyperparameter in linearSVC and Logistic Regression model

In logistic regression, the penalty hyperparameter determines regularization type ('l1' or 'l2'). 'l1' penalizes absolute coefficient values, promoting sparsity, while 'l2' penalizes squared coefficients, leading to smoother boundaries. Adjusting this hyperparameter controls overfitting, fostering simpler models for improved generalization. In LinearSVC, the penalty hyperparameter controls regularization ('l1' or 'l2'). 'l1' induces sparsity by penalizing absolute coefficients, resulting in sparse solutions. 'l2' penalizes squared coefficients, promoting smaller values and smoother boundaries. Adjusting this hyperparameter regulates model complexity, mitigating overfitting and enhancing generalization to new data.

We made the observations that are mentioned below using the hyperparameters. as C=1, tolerance = 1e-4, loss = squared hinge and dual = False LinearSVC.

Table 4: Effect of varying penalty type on LinearSVC and LogisticRegression (5 trials each)

Classifier	Penalty	Avg Training Time (s)	Avg Test Accuracy
LinearSVC	l1	1.9115	0.9610
	l2	0.1040	0.9537
LogisticRegression	l1	1.8866	0.9806
	l2	0.0230	0.9587

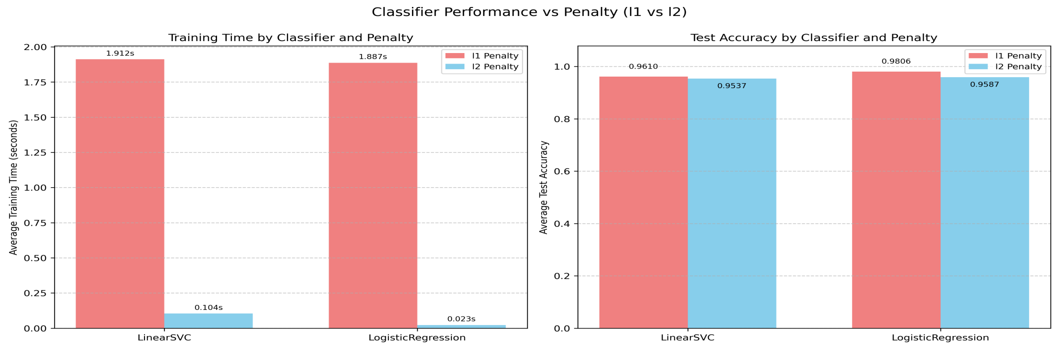

Figure 5: Classifier Performance vs Penalty (l1 vs l2).