CS771 Assignment 1

Adarsh Kumar
210048
Dept. of Civil Engineering
adarshkum21@iitk.ac.in

Aryan
210198
Dept. of Chemical Engineering
aryan21@iitk.ac.in

Narottam Kumar Pankaj
210652
Dept. of Chemical Engineering
narottamkp21@iitk.ac.in

Paras
210698
Dept. of Chemistry
paras21@iitk.ac.in

Rajat Phogat
210814
Dept. of Chemical Engineering
rajatp21@iitk.ac.in

Sumit Saurabh
211075
Dept. of Mechanical Engineering
sumits21@iitk.ac.in

Umesh Rathore
211125
Dept. of Mechanical Engineering
umeshr21@iitk.ac.in

Abstract

This document describes the methodology and approach used by our group in the Assignment-1 of the course CS771: An Introduction to Machine Learning, offered at IITK in the semester 2023-24-Summer.

1 Linear model to predict the time it takes for the upper signal to reach the finish line for Arbiter PUF

1.1 Arbiter PUFs – A Brief Introduction

Arbiter Physically Unclonable Functions (PUFs) are hardware mechanisms that leverage unpredictable variations in data transmission speeds across different instances of the same design to enhance security. These systems are composed of a series of multiplexers (hereafter referred to as muxes), each controlled by a selection bit. A specific configuration of these selection bits constitutes a "challenge" or "question," and the outcome is determined by which signal reaches the end of the PUF first, serving as the "answer." This answer is unique to the particular hardware of the PUF, making each response distinct for every device.

1.2 Mathematical Derivation

For a single Arbiter Puff with 32 challenges, we have:

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

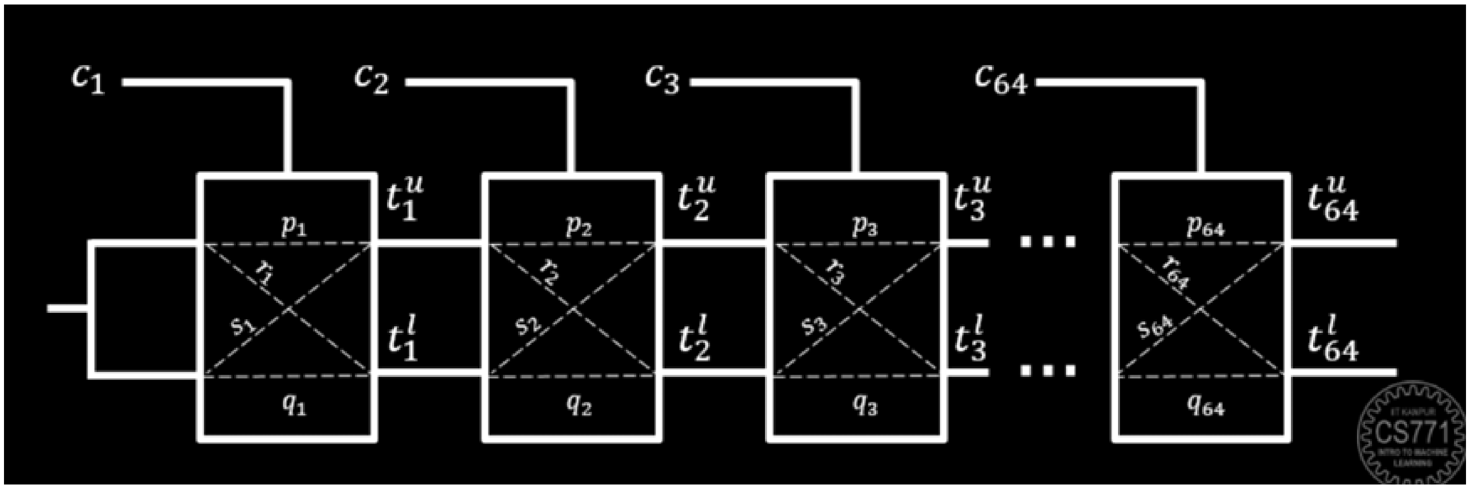

Figure 1: A Simple Arbiter PUF with 64 multiplexers

Where:

$$x_i = (1-2c_i)\cdot(1-2c_{i+1})\cdots(1-2c_{31})$$
$$w_0 = \alpha_0$$
$$w_i = \alpha_i + \beta_{i-1} \quad (\text{ for } i>0)$$
$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}$$
$$\beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$
$$b = \beta_{31}$$

Where p_i, q_i, r_i, and s_i account for the losses in the i^{th} mux of the arbiter puff. This implies $\Delta_i = (1-2c_i)\,\Delta_{i-1} + \alpha_i\,(1-2c_i) + \beta_i$

$$\Delta_{-1} = 0$$

If $\Delta_{31} < 0$, upper signal wins and the answer is 0 . If $\Delta_{31} > 0$, lower signal wins and the answer is 1 . Thus, the answer is simply $\left(\operatorname{sign}\left(w^\top x + b\right) + 1\right)/2$.

Given 2 arbiter puffs in a single CAR-PUF such that: $\Delta_{u_{31}} = u^\top x + p$ for one puff and $\Delta_{v_{31}} = v^\top x + q$ for the other This implies, from equation 1 :

$$u_i = \alpha_{u_i} + \beta_{u_{i-1}} \quad (\text{ for } i>0)$$
$$v_i = \alpha_{v_i} + \beta_{v_{i-1}} \quad (\text{ for } i>0)$$

Defining $w_i = u_i - v_i$, this implies: $w_i = \left(\beta_{u_{i-1}} - \beta_{v_{i-1}}\right) - \left(\alpha_{u_i} - \alpha_{v_i}\right)$ Therefore $\Delta_{u_{31}} - \Delta_{v_{31}} = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$ such that $\mathbf{w} = \mathbf{u} - \mathbf{v}$ and $b = p - q$ If $|\Delta_{u_{31}} - \Delta_{v_{31}}| \leq \tau$, this implies the answer is 0 ; else, the answer is 1 . This implies $|\Delta_{u_{31}} - \Delta_{v_{31}}| = \tau$ is the Decision Boundary. This implies $\left|\mathbf{w}^{\mathrm{T}}\mathbf{x} + b\right| = \tau$ is the Decision Boundary Squaring on both sides we get: $\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b\right)^2 = \tau^2$. If $\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b\right)^2 \leq \tau^2$, then the answer is 0 , else the answer is 1 .

Expanding $\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b\right)^2 = \tau^2$:

$$\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)^2 + 2b\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) + b^2 = \tau^2$$

Further Expanding:

$$\begin{aligned}&\left(w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_{31}x_{31}\right)^2\\ &+ 2b\left(w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_{31}x_{31}\right) + b^2 = \tau^2\end{aligned}$$

This implies:

$$\begin{aligned}
&(w_0x_0)^2 + (w_1x_1)^2 + (w_2x_2)^2 + \cdots + (w_{31}x_{31})^2 \\
&+ 2w_0x_0\left(w_1x_1 + w_2x_2 + \cdots + w_{31}x_{31}\right) \\
&+ 2w_1x_1\left(w_2x_2 + w_3x_3 + \cdots + w_{31}x_{31}\right) \\
&+ 2w_2x_2\left(w_3x_3 + w_4x_4 + \cdots + w_{31}x_{31}\right) \\
&+ \cdots \\
&+ 2w_{30}x_{30}\left(w_{31}x_{31}\right) \\
&+ 2bw_0x_0 + 2bw_1x_1 + 2bw_2x_2 + 2bw_3x_3 + \cdots + 2bw_{31}x_{31} + b^2 - \tau^2 = 0
\end{aligned}$$

Therefore finally:

$$\begin{aligned}
&w_0^2x_0^2 + w_1^2x_1^2 + w_2^2x_2^2 + \cdots + w_{31}^2x_{31}^2 \\
&+ 2w_0w_1x_0x_1 + 2w_0w_2x_0x_2 + 2w_0w_3x_0x_3 + \cdots + 2w_0w_{31}x_0x_{31} \\
&+ 2w_1w_2x_1x_2 + 2w_1w_3x_1x_3 + 2w_1w_4x_1x_4 + \cdots + 2w_1w_{31}x_1x_{31} \\
&+ 2w_2w_3x_2x_3 + 2w_2w_4x_2x_4 + 2w_2w_5x_2x_5 + \cdots + 2w_2w_{31}x_2x_{31} \\
&+ \cdots \\
&+ 2w_{30}w_{31}x_{30}x_{31} \\
&+ 2bw_0x_0 + 2bw_1x_1 + 2bw_2x_2 + 2bw_3x_3 + \cdots + 2bw_{31}x_{31} + b^2 - \tau^2 = 0
\end{aligned}$$

$\mathrm{x}_i = \prod_{j=i}^{31}(1-2c_j) \in \{-1, 1\}$ because $c_j \in \{0, 1\}$.

This implies ${x_i}^2 = 1$ always. So,

$$w_0^2x_0^2 + w_1^2x_1^2 + w_2^2x_2^2 + \cdots + w_{31}^2x_{31}^2 = \sum_{i=0}^{31} w_i^2$$

Therefore, $\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b\right)^2 - \tau^2$ can be written as: $\left(\mathbf{W}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{c}) + b'\right)$ Where:

$$\mathbf{W} = \begin{bmatrix} w_0w_1 \\ \vdots \\ w_0w_{31} \\ w_1w_2 \\ \vdots \\ w_1w_{31} \\ w_2w_3 \\ \vdots \\ w_2w_{31} \\ (p-q)w_0 \\ \vdots \\ (p-q)w_{31} \end{bmatrix} \quad \boldsymbol{\alpha} = \begin{bmatrix} 2x_0x_1 \\ \vdots \\ 2x_0x_{31} \\ 2x_1x_2 \\ \vdots \\ 2x_1x_{31} \\ 2x_2x_3 \\ \vdots \\ 2x_2x_{31} \\ 2x_0 \\ \vdots \\ 2x_{31} \end{bmatrix} \quad b' = (p-q)^2 - \tau^2 + \sum_{i=0}^{31} w_i^2$$

$$x_i = \prod_{j=i}^{31}(1-2c_j)$$

This implies the answer is nothing but:

$$\left(\operatorname{sign}\left(W^T\emptyset(c) + b'\right) + 1\right)/2$$

With the dimensions of $\mathbf{W}$ being $\left(\binom{32}{2} + 32, 1\right) = (528, 1)$

If $\left(W^T\emptyset(c) + b'\right) \geq 0$, then $\operatorname{sign}\left(W^T\emptyset(c) + b'\right) = 1$, and the answer is $(1+1)/2 = 1$. If $\left(W^T\emptyset(c) + b'\right) \leq 0$, then $\operatorname{sign}\left(W^T\emptyset(c) + b'\right) = -1$, and the answer is $(-1+1)/2 = 0$.

2 Dimensionality of the model W

From previous part,

$$\mathbf{W} = \begin{bmatrix} w_0 w_1 \\ \vdots \\ w_0 w_{31} \\ w_1 w_2 \\ \vdots \\ w_1 w_{31} \\ w_2 w_3 \\ \vdots \\ w_2 w_{31} \\ (p-q) w_0 \\ \vdots \\ (p-q) w_{31} \end{bmatrix}$$

Clearly, it is 528 x 1 matrix. Hence, dimensionility of the **W**(model) is **528**.

3 Linear models to predict Response0 and Response1

3.1 Another type of PUF — COCO-PUF

A COCO-PUF utilizes two arbiter PUFs, PUF0 and PUF1, each equipped with its own set of multiplexers, which may exhibit different delays. When a challenge is presented, it is fed into both PUFs. Instead of having the lower and upper signals within PUF0 compete with each other, the lower signal from PUF0 competes with the lower signal from PUF1 using an arbiter referred to as Arbiter0. This competition generates a response known as Response0. Similarly, the upper signal from PUF0 competes with the upper signal from PUF1 using another arbiter, Arbiter1, which produces a response called Response1.

3.2 Using ML to crack COCO-PUF

3.2.1 COCO PUF Overview

- **COCO-PUF:** Combines two arbiter PUFs (PUF0 and PUF1) that receive the same challenge but may have different delay characteristics.
- **Responses:**
  - $r_0(c)$: Output of PUF0.
  - $r_1(c)$: Output of PUF1.

3.2.2 Linear Model for Response Prediction

- The responses are predicted using a linear model:

$$r_i(c) = \frac{1 + \text{sign}\left(\tilde{W}_i^T \tilde{\phi}(c) + \tilde{b}_i\right)}{2}$$

- $i = 0, 1$ (for PUF0 and PUF1).

3.2.3 Feature Mapping

- Mapping $\tilde{\phi} : \{0,1\}^{32} \to \mathbb{R}^{\tilde{D}}$:
  - Transforms the 32-bit challenge vector into a higher-dimensional space.

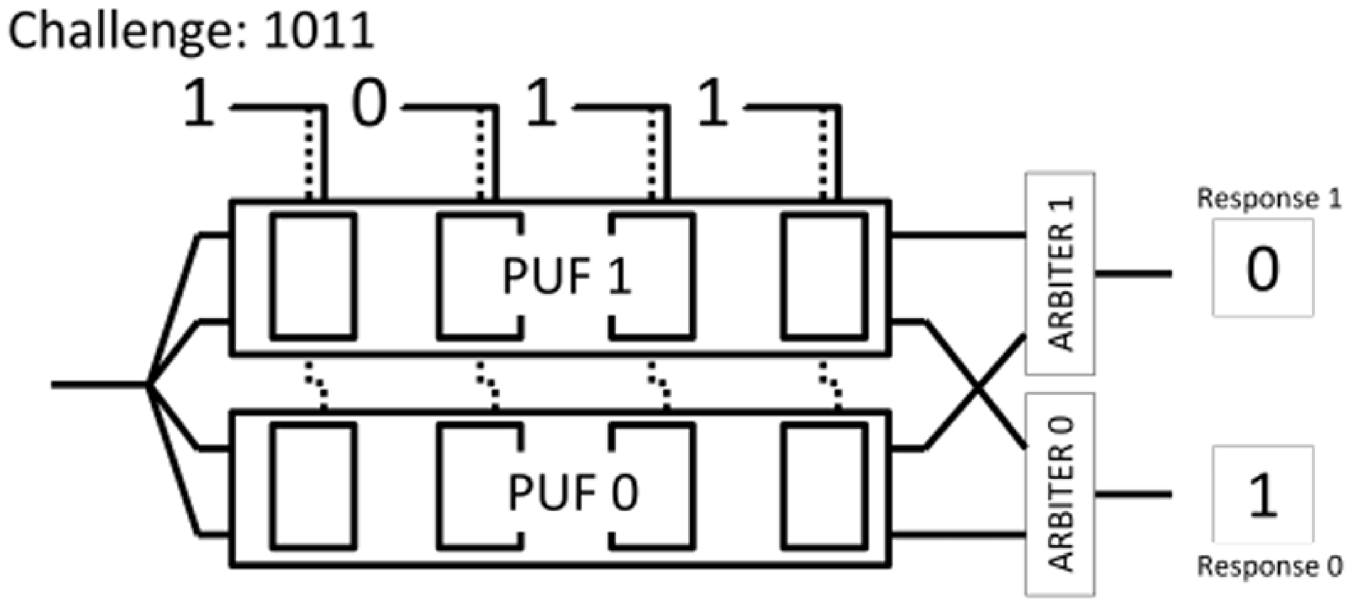

Figure 2: A COCO-PUF with 4-bit challengers and 2-bit responses

- Example mapping:

$$\tilde{\phi}(c) = [1, c_1, c_2, \ldots, c_{32}, c_1c_2, c_1c_3, \ldots, c_{31}c_{32}]$$

  - Bias term: 1
  - Linear terms: $c_1, c_2, \ldots, c_{32}$
  - Interaction terms: c_ic_j for all $i < j$

3.2.4 Linear Model Parameters

- Weight Vector $\tilde{W}_i \in \mathbb{R}^{\tilde{D}}$: Contribution of each feature.
- Bias Term $\tilde{b}_i \in \mathbb{R}$: Accounts for constant offsets.

3.2.5 Response Predictions

1. Response0 Prediction:
   - Use the mapping $\tilde{\phi}(c)$ to transform the challenge.
   - Apply the linear model:

$$r_0(c) = \frac{1 + \text{sign}\left(\tilde{W}_0^T \tilde{\phi}(c) + \tilde{b}_0\right)}{2}$$

2. Response1 Prediction:
   - Use the same mapping $\tilde{\phi}(c)$.
   - Apply the linear model:

$$r_1(c) = \frac{1 + \text{sign}\left(\tilde{W}_1^T \tilde{\phi}(c) + \tilde{b}_1\right)}{2}$$

3.2.6 Dimensionality

- For 32 bits:

$$\tilde{D} = 32 + \frac{32 \cdot 31}{2} = 528$$

3.2.7 Summary

- Feature Mapping $\tilde{\phi}(c)$: Captures relationships between challenge bits.
- Linear Models $\tilde{W}_i, \tilde{b}_i$: Used to predict responses based on mapped features.

This approach allows prediction of responses $r_0(c)$ and $r_1(c)$ for any challenge c using a linear classifier model, without using PUF-specific delay constants directly in the feature mapping.

4 Dimensionality of the linear models W_0 and W_1

From previous part,

$$\widetilde{W}_0 = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_1 \\ \delta_2 + \gamma_1 \\ \vdots \\ \delta_{31} + \gamma_{30} + \beta_{31} \end{bmatrix} \text{ and } \widetilde{W}_1 = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_1 \\ \delta_2 + \gamma_1 \\ \vdots \\ \delta_{31} + \gamma_{30} + \beta_{31} \end{bmatrix}$$

Note: Here the γ_i and δ_i for these two models are slightly different.
Clearly, both $\widetilde{W}_0$ and $\widetilde{W}_1$ are 528×1 matrices.
Hence, the dimensionality for both the $\widetilde{W}_0$ and $\widetilde{W}_1$ is 528×1.

5 Code for the two linear models W_0, b_0, W_1, b_1

Code Submitted

6 Outcomes of the Experiments

6.1 Training Set

The model is trained using the dataset in `public_trn.txt` and tested with the dataset in `public_tst.txt`, employing a different set of hyperparameters. The accuracy and training time are then recorded.

1. **Effect of changing the loss hyperparameter in LinearSVC**
The *loss function* characterizes how well the model performs on a given dataset. `sklearn.svm.LinearSVC` provides two loss functions, namely: *hinge* and *squared hinge*. We made the observations that are mentioned below using the hyperparameters $C = 1$, `tolerance` $= 1e - 3$ `and dual = True for LinearSVC.`

Table 1: Loss Hyperparameters

Loss	Training Time(in sec)	Accuracy y0	Accuracy y1
Hinge	19.12	0.9714	0.9922
Squared Hinge	15.9	0.9781	0.9916

2. **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 C values correspond to stronger regularization. In both Logistic Regression and Linear SVC, adjusting C affects how closely the model fits the training data. Increasing C results in a tighter fit, which may lead to overfitting, while decreasing C promotes simpler decision boundaries, aiding generalization but risking underfitting.

We made the observations that are mentioned below using the hyperparameters `loss` = `Squared Hinge`, `tolerance` $= 1e-3$, `penalty` = `L2` `and` `dual` = `False` for **LinearSVC** and `tolerance` $= 1e-3$, `penalty` = `L2` `and` `dual` = `False` for **Logistic Regression:**

Table 2: C Hyperparameters in LinearSVC

Cvalue	Training Time(in sec)	Accuracy y0	Accuracy y1
0.01	0.9739	0.987	5.35
0.1	0.974	0.9913	5.32
1	0.9739	0.9922	5.48
10	0.9737	0.9919	5.85
100	0.9737	0.9921	5.85

Table 3: C Hyperparameters in Logistic Regression

Cvalue	Training Time(in sec)	Accuracy y0	Accuracy y1
0.01	0.9729	0.9818	3.53
0.1	0.9743	0.9876	3.69
1	0.9739	0.9919	4.35
10	0.9739	0.9928	4.84
100	0.9739	0.9921	5.43

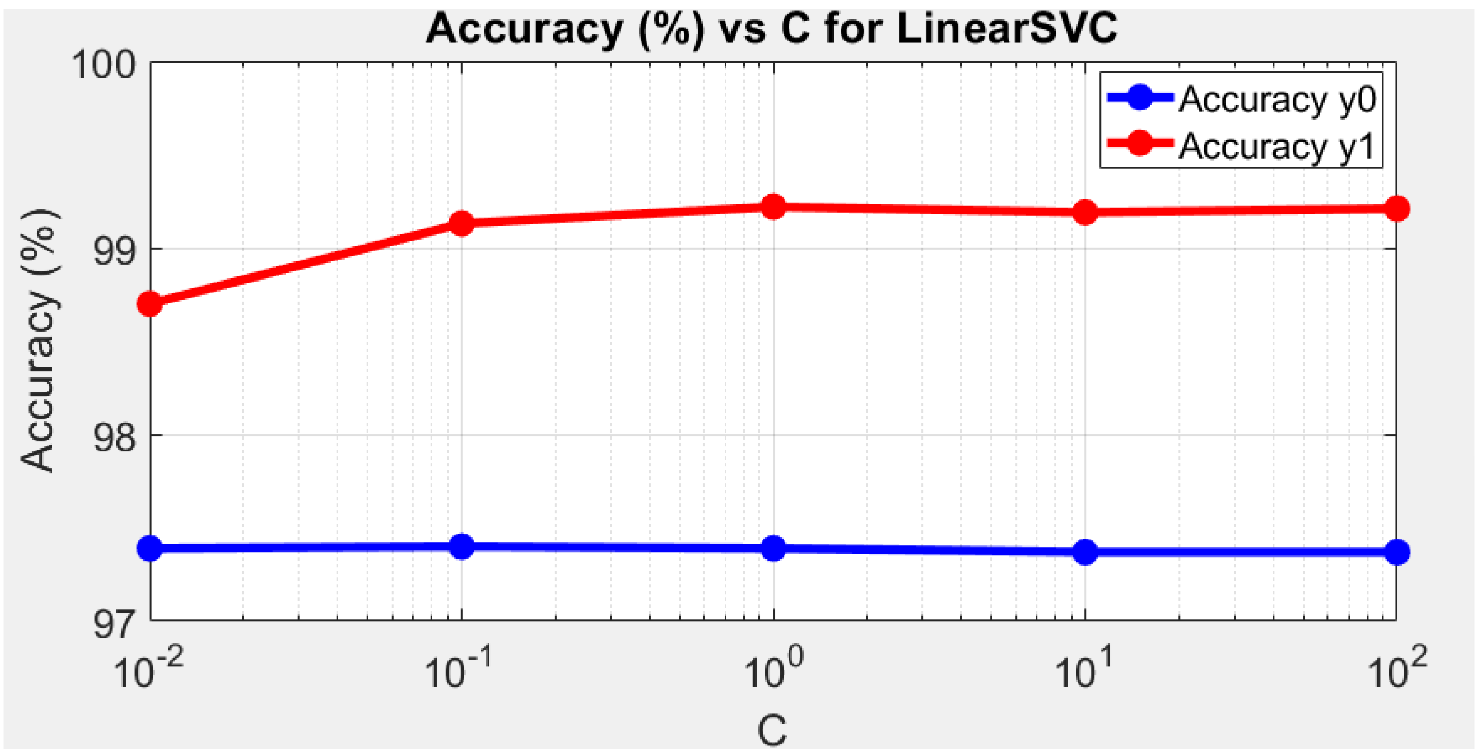

Figure 3: Accuracy(%) vs C for LinearSVC

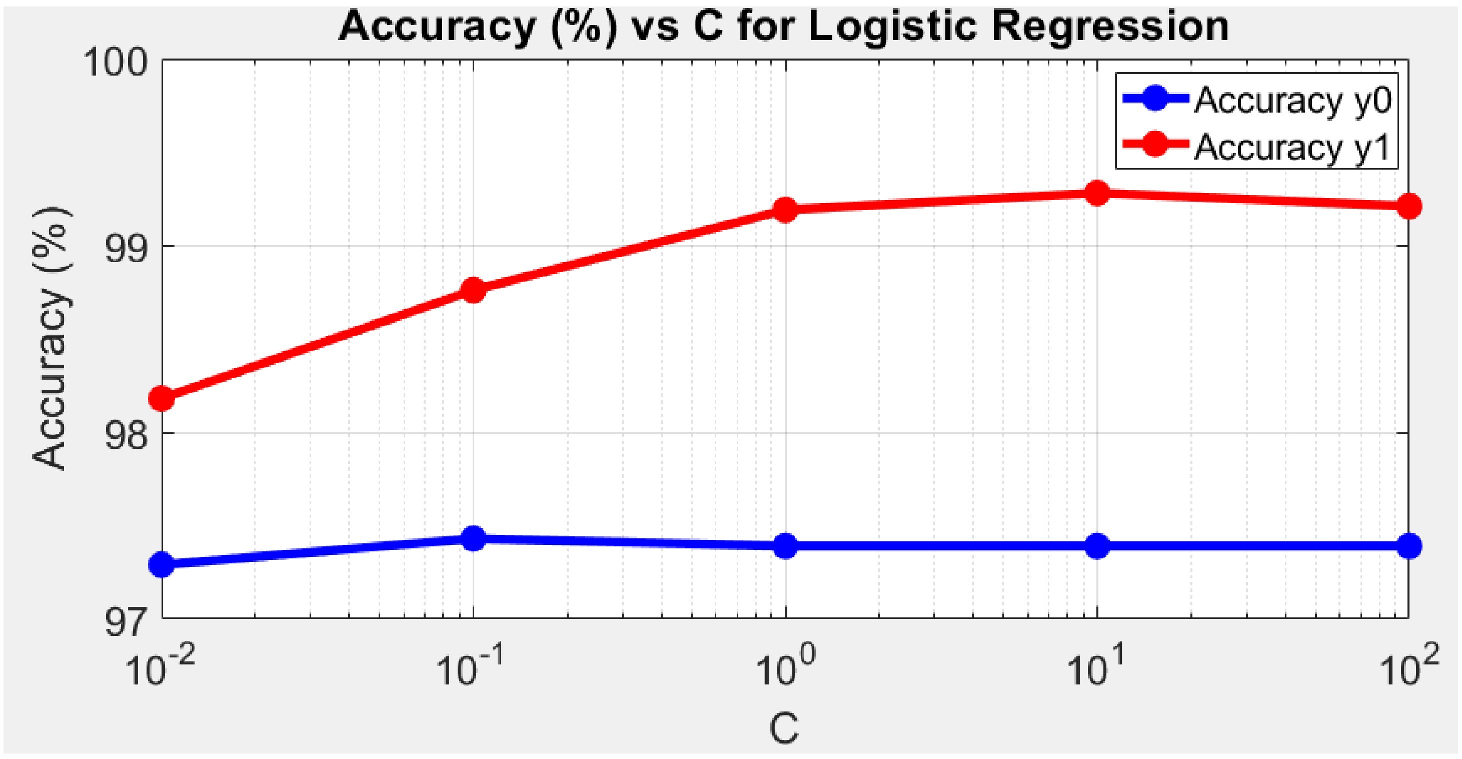

Figure 4: Accuracy(%) vs C for Logistic Regression

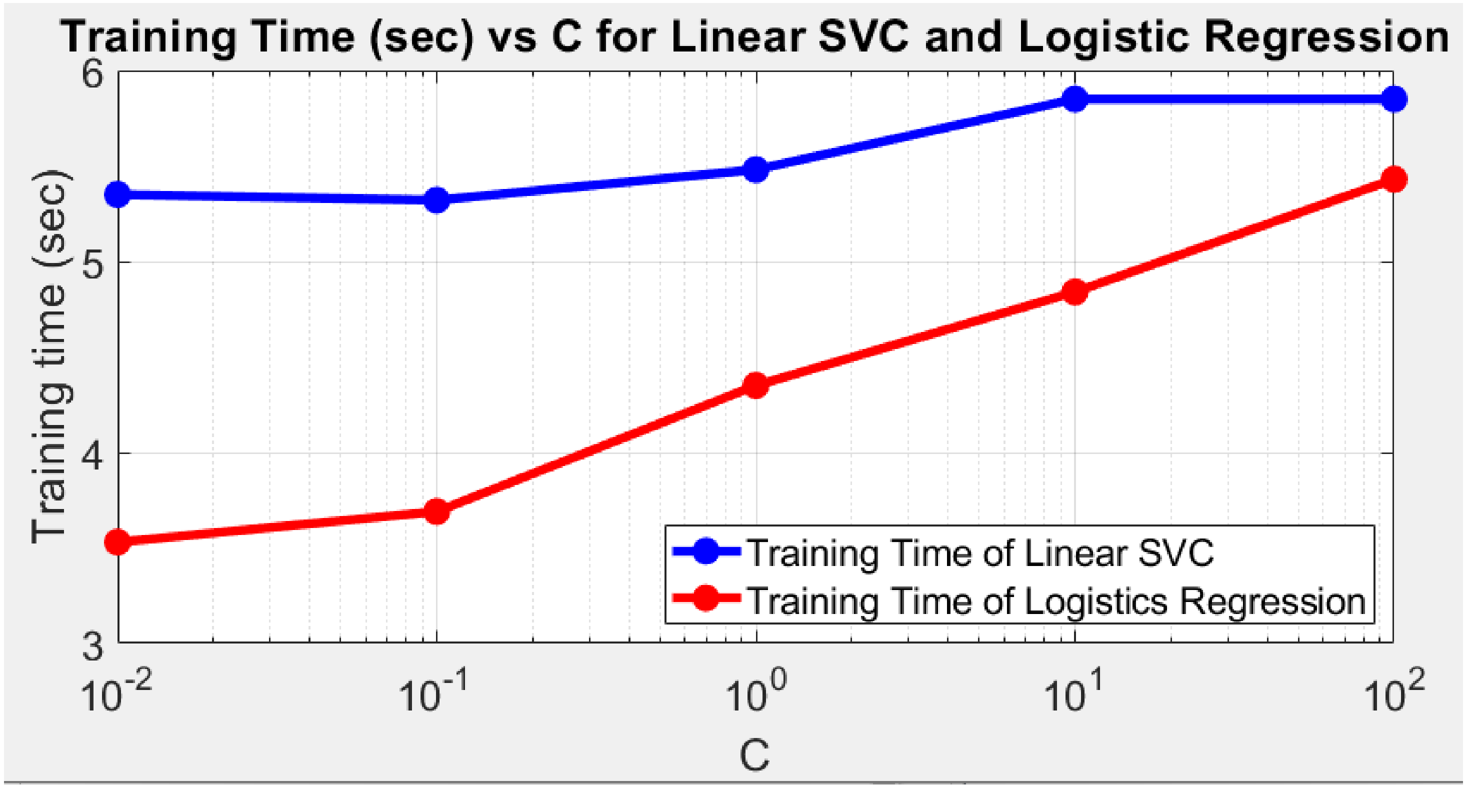

Figure 5: Training Time(sec) vs C for LinearSVC and Logistic Regression

3. **Effect of changing the *tol* hyperparameter in LinearSVC and Logistic Regression model**

The *tol* hyperparameter, which stands for tolerance, establishes the convergence threshold for optimization algorithms in LinearSVC and LogisticRegression. It dictates when the iterative optimization process should stop by assessing the change in the objective function or coefficients between iterations. A smaller tol value means stricter convergence criteria, leading to longer training times but potentially enhancing accuracy. On the other hand, a larger tol speeds up the training process but might reduce precision. Therefore, selecting an appropriate tol value is crucial for balancing computational efficiency and model accuracy.

We made the observations that are mentioned below using the hyperparameters loss = Squared Hinge, C=10, penalty = L2 and dual = False for LinearSVC and C = 100, penalty = L2, solver = 'lbfgs' and dual = False for Logistic Regression:

Table 4: tol Hyperparameters in LinearSVC

tol value	Training Time(in sec)	Accuracy y0	Accuracy y1
0.1	6.24	0.9693	0.9793
0.01	7.5	0.9751	0.9915
0.001	9.3	0.9737	0.9919
0.0001	11.69	0.9738	0.992
0.00001	13.1	0.9738	0.9921

Table 5: tol Hyperparameters in Logistic Regression

C value	Training Time(in sec)	Accuracy y0	Accuracy y1
0.1	7.27	0.9739	0.9921
0.01	7.47	0.9739	0.9921
0.001	7.85	0.9739	0.9921
0.0001	7.94	0.9739	0.9921
0.00001	8.89	0.9739	0.9921

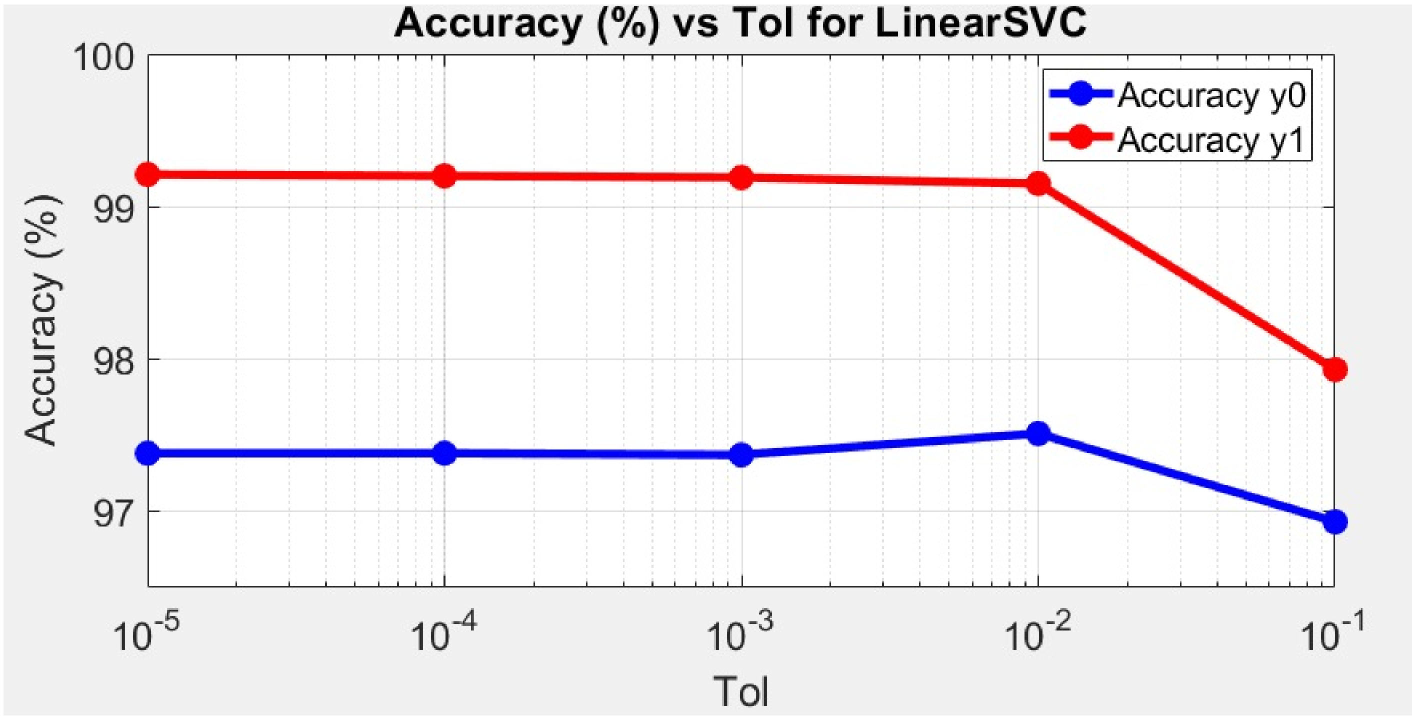

Figure 6: Accuracy(%) vs tol for LinearSVC

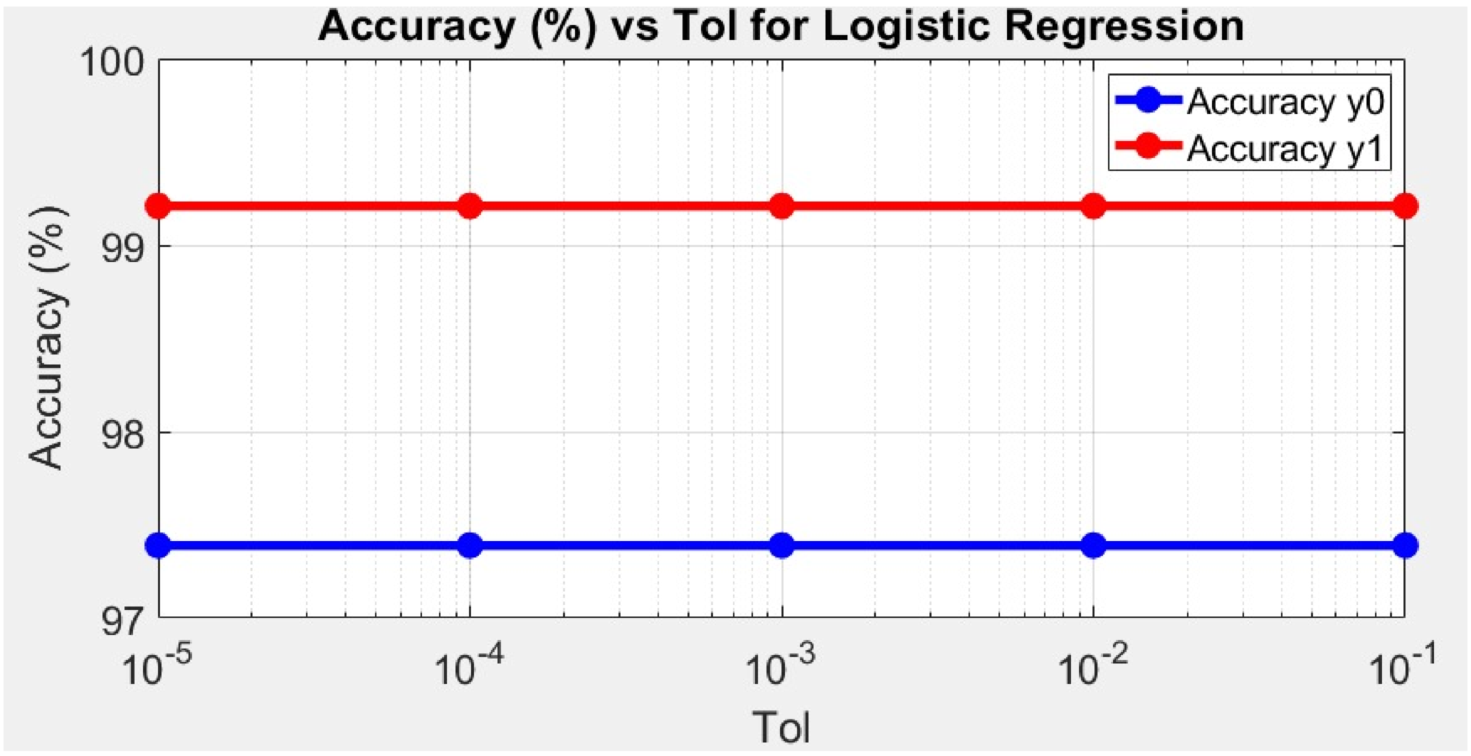

Figure 7: Accuracy(%) vs tol for Logistic Regression

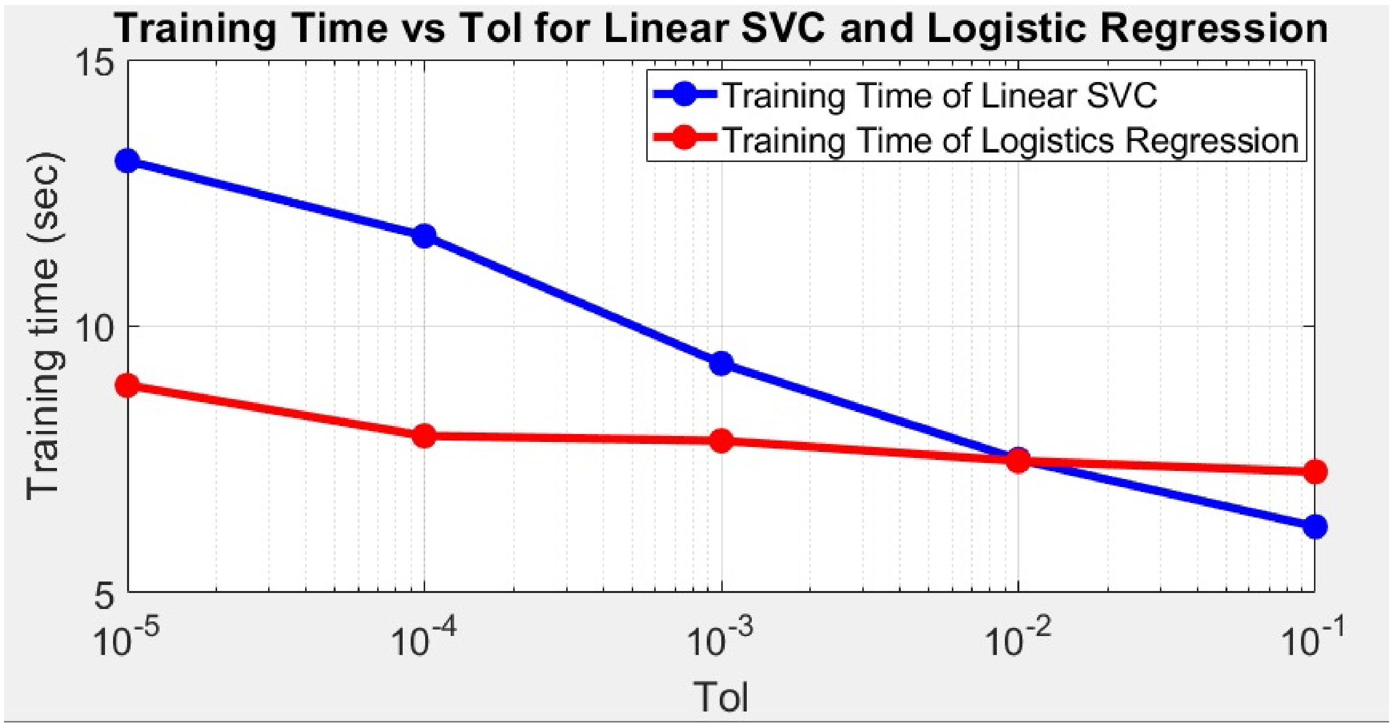

Figure 8: Training Time(sec) vs tol for LinearSVC and Logistic Regression