CS771 : Introduction to Machine Learning
Assignment - 1
Team: Tensor Titans

Raghav Manglik
220854
Dept. of Computer Science and Engineering
IIT Kanpur
`mraghav22@iitk.ac.in`

Srishti Chandra
221088
Dept. of Computer Science and Engineering
IIT Kanpur
`srishtic22@iitk.ac.in`

Raghav Madan
220853
Department of Electrical Engineering
IIT Kanpur
`raghavm22@iitk.ac.in`

Pathe Nevish Ashok
220757
Dept. of Computer Science and Engineering
IIT Kanpur
`pathena22@iitk.ac.in`

Apoorv Tandon
220192
Dept. of Computer Science and Engineering
IIT Kanpur
`apoorvt22@iitk.ac.in`

Khushi Gupta
220531
Dept. of Computer Science and Engineering
IIT Kanpur
`khushig22@iitk.ac.in`

Abstract

This is our solution to the first Assignment of the course CS771. We are required to show (with a mathematical derivation) how a Companion Arbiter PUF can be broken by a single linear model, then report the outcomes of our models, along with the effects of hyperparameter tuning on model training time and accuracy.

Preprint. Under review.

1 Part 1

1.1 Arbiter PUFs – A Brief Introduction

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 "question/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.

1.2 Using ML to crack Arbiter PUFs

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

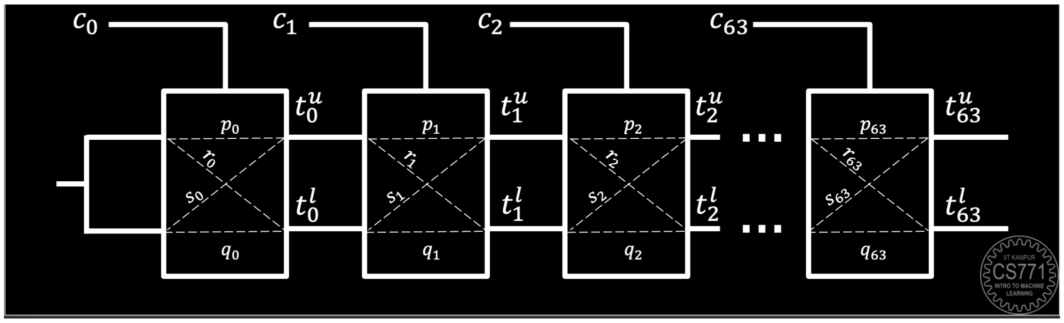

Figure 1: Analysis – Arbiter PUF

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

$$t_1^u = (1 - c_1).(t_0^u + p_1) + c_1.(t_0^l + s_1)$$
$$t_1^l = (1 - c_1).(t_0^l + q_1) + c_1.(t_0^u + r_1)$$
$$\Delta_i = t_i^u - t_i^l$$

$$\begin{aligned}\Delta_1 &= (1 - c_1) \cdot (t_0^u + p_1 - t_0^l - q_1) + c_1 \cdot (t_0^l + s_1 - t_0^u - r_1) \\ &= (1 - c_1) \cdot (\Delta_0 + p_1 - q_1) + c_1 \cdot (-\Delta_0 + s_1 - r_1) \\ &= (1 - 2c_1) \cdot \Delta_0 + (q_1 - p_1 + s_1 - r_1) \cdot c_1 + (p_1 - q_1)\end{aligned}$$

$$\text{Let } d_i = (1 - 2c_i)$$

$$\Delta_1 = \Delta_0 \cdot d_1 + \alpha_1 \cdot d_1 + \beta_1$$
$$\alpha_1 = \frac{p_1 - q_1 + r_1 - s_1}{2}, \beta_1 = \frac{p_1 - q_1 - r_1 + s_1}{2}$$

A similar relation holds for any stage i:

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

Let $\Delta_{-1} = 0$ (absorb initial delays into p_0, q_0, r_0, s_0)
We can keep going recursively:

$$\Delta_0 = \alpha_0 \cdot d_0 + \beta_0 \quad (\text{since } \Delta_{-1} = 0)$$

$$\Delta_1 = \Delta_0 \cdot d_1 + \alpha_1 \cdot d_1 + \beta_1$$

plugging in the value of Δ_0, we get

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

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

noticing the pattern,

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

where

$$x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31}$$

$$w_0 = \alpha_0$$

$$w_i = \alpha_i + \beta_{i-1} \quad \text{for } i > 0$$

This gives us the final difference in timings experienced between both the signals for a PUF, Δ_{31} as an affine function, $\mathbf{w}^\top \mathbf{x} + b$

1.3 Using ML to crack Companion Arbiter PUF (CAR-PUF)

A CAR-PUF uses 2 arbiter PUFs – a working PUF and a reference PUF, as well as a secret threshold value $\tau > 0$.
Given a challenge, it is fed into both the working PUF and reference PUF and the timings for the upper and lower signals for both PUFs are measured. Let Δ_w, Δ_r be the difference in timings experienced for the two PUFs on the same challenge.
The response to this challenge is 0 if $|\Delta_w - \Delta_r| \leq \tau$ and the response is 1 if $|\Delta_w - \Delta_r| > \tau$ where $\tau > 0$ is the secret threshold value.

So in this section, let us show that the CAR-PUF can be cracked using a Linear ML model. According to the question, the CAR-PUF which we have been given is a CAR-PUF with 32 bit challenges. So we will show our analysis on a 32 bit CAR-PUF. Analysis for CAR-PUFs with higher bits of challenges follow similarly to the analysis below.

In the previous section we showed that delay after any stage i for an arbiter PUF is given by :

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

where

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

The delay at final stage is given by (assuming a 32 bit arbiter PUF) is given by

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

where $x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31}$, $w_0 = \alpha_0$ and $w_i = \alpha_i + \beta_{i-1}$ for $i > 0$.

Note to the reader Please observe that we have used 0-based indexing, that is the first multiplexer of the PUF is numbered 0 and not 1, hence the delay associated with it is Δ_0 and the challenge associated with it is c_0.

As a CAR-PUF is nothing but just 2 arbiter PUFs working together, the delay at final stage of the working arbiter PUF and reference arbiter PUF are respectively,

Working arbiter PUF

$$\Delta_{31^w} = w_0^w \cdot x_0 + w_1^w \cdot x_1 + \ldots + w_{31}^w \cdot x_{31} + \beta_{31}^w = \mathbf{W_w}^\top \mathbf{X} + b^w$$

where $x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31}$, $w_0^w = \alpha_0^w$, $w_i^w = \alpha_i^w + \beta_{i-1}^w$ for $i > 0$ and $b^w = \beta_{31}^w$

Reference arbiter PUF

$$\Delta_{31^r} = w_0^r \cdot x_0 + w_1^r \cdot x_1 + \ldots + w_{31}^r \cdot x_{31} + \beta_{31}^r = \mathbf{W_r}^\top \mathbf{X} + b^r$$

where $x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31}$, $w_0^r = \alpha_0^r$, $w_i^r = \alpha_i^r + \beta_{i-1}^r$ for $i > 0$ and $b^r = \beta_{31}^r$

Observe that $x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31}$ is same for both the reference arbiter PUF as well as the working arbiter PUF. This is because at each stage the challenge fed to the working arbiter PUF and reference arbiter PUF is the same and $d_i = 1 - 2c_i$, so d_i for both kind of PUFs is the same and hence x_i is same.

We have,

$$\begin{aligned}\Delta_{31^w} - \Delta_{31^r} &= (w_0^w - w_0^r) \cdot x_0 + (w_1^w - w_1^r) \cdot x_1 + \ldots + (w_{31}^w - w_{31}^r) \cdot x_{31} + \beta_{31}^w - \beta_{31}^r \\ &= w_0 \cdot x_0 + w_1 \cdot x_1 + \ldots + w_{31} \cdot x_{31} + b \\ &= \mathbf{W}^\top \mathbf{X} + b\end{aligned}$$

where $w_i = w_i^w - w_i^r$ and $b = b^w - b^r$ and $\mathbf{W}$ is 32 dimensional vector containing w_i.
The response to the given set of challenges is 0 if $|\Delta_{31^w} - \Delta_{31^r}| \leq \tau$ and the response is 1 if $|\Delta_{31^w} - \Delta_{31^r}| > \tau$ where $\tau > 0$ is the secret threshold value.

This statement is same as saying that *"The response to the given set of challenges is* 0 *if* $(\mathbf{W}^\top \mathbf{X} + b)^2 \leq \tau^2$ *and the response is* 1 *if* $(\mathbf{W}^\top \mathbf{X} + b)^2 > \tau^2$ *where* $\tau > 0$ *is the secret threshold value."*
(We just squared both sides of the inequality and replaced $(\Delta_{31^w} - \Delta_{31^r})$ *with* $\mathbf{W}^\top \mathbf{X} + b$*).*

Therefore the response is

$$\boxed{\frac{\text{sign}((\mathbf{W}^\top \mathbf{X} + b)^2 - \tau^2) + 1}{2}}$$

Now our task is to convert this into a linear model.

$$\begin{aligned}(\mathbf{W}^\top \mathbf{X} + b)^2 - \tau^2 &= (w_0 \cdot x_0 + w_1 \cdot x_1 + \ldots + w_{31} \cdot x_{31} + b)^2 - \tau^2 \\ &= \sum_{i=0}^{31} (w_i \cdot x_i)^2 + 2b \cdot \sum_{i=0}^{31} w_i \cdot x_i + 2 \cdot \sum_{i=0}^{31} \sum_{j=i+1}^{31} w_i \cdot x_i \cdot w_j \cdot x_j + b^2 - \tau^2\end{aligned}$$

Observe that $\sum_{i=0}^{31} (w_i \cdot x_i)^2 = \sum_{i=0}^{31} (w_i)^2$ since $x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31} = \pm 1$, so $x_i^2 = 1$ always.
Note : $x_i = \pm 1$ because $d_i = 1 - 2c_i$ and since c_i (1 bit of the challenge string) is either 0 or 1,

$d_i = \pm 1$. Therefore $x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31} = \pm 1$. So,

$$\begin{aligned}(\mathbf{W}^\top \mathbf{X} + b)^2 - \tau^2 &= \sum_{i=0}^{31} (w_i)^2 + 2b \cdot \sum_{i=0}^{31} w_i \cdot x_i + 2 \cdot \sum_{i=0}^{31} \sum_{j=i+1}^{31} w_i \cdot x_i \cdot w_j \cdot x_j + b^2 - \tau^2 \\ &= 2b \cdot \sum_{i=0}^{31} w_i \cdot x_i + 2 \cdot \sum_{i=0}^{31} \sum_{j=i+1}^{31} (w_i \cdot w_j) \cdot x_i \cdot x_j + b^2 + \sum_{i=0}^{31} w_i^2 - \tau^2 \\ &= \mathbf{W_f}^\top \mathbf{X_f} + b_f\end{aligned}$$

where, $\mathbf{W_f} = [2b \cdot w_0 \quad 2b \cdot w_1 \quad \cdots \quad 2b \cdot w_{31} \quad 2 \cdot w_0 \cdot w_1 \quad \cdots \quad 2 \cdot w_0 \cdot w_{31} \quad \cdots \quad \cdots 2 \cdot w_{30} \cdot w_{31}]^\top$
$\mathbf{X_f} = [x_0 \quad x_1 \quad \cdots \quad x_{31} \quad x_0 \cdot x_1 \quad \cdots \quad x_0 \cdot x_{31} \quad \cdots \quad \cdots x_{30} \cdot x_{31}]^\top$
and $b_f = b^2 + \sum_{i=0}^{31} w_i^2 - \tau^2$.

$\mathbf{X_f}$ is a $\binom{32}{2} + 32 = \binom{33}{2} = 528$ dimensional feature vector, since there are $\binom{32}{2}$ possible $x_i \cdot x_j$ such that $0 \le i < j \le 31$ and 32 different x_i for $0 \le i \le 31$.

The vector $\mathbf{X_f}$ is equivalent to the map $\phi(\mathbf{c}) : \{0,1\}^{32} \rightarrow \{-1,1\}^{528}$ which maps the 32-bit 0/1-valued challenge vectors to 528-dimensional feature vectors with output as $\{-1,1\}$ since $x_i = \pm 1$ and $x_i \cdot x_j = \pm 1$.

This concludes us with the fact that the map $\phi : \{0,1\}^{32} \rightarrow \{-1,1\}^{528}$ maps the 32-bit 0/1-valued challenge vectors to 528-dimensional feature vectors so that for any CAR-PUF, there exists a 528-dimensional linear model $\mathbf{W_f} \in \mathbb{R}^{528}$ and a bias term $b_f \in \mathbb{R}$ such that for all CRPs($\mathbf{c}$,r) with $\mathbf{c} \in \{0,1\}^{32}, r \in \{0,1\}$, we have

$$\boxed{r = \frac{1 + \text{sign}(\mathbf{W_f}^\top \phi(\mathbf{c}) + b_f)}{2}}$$

2 Part 2

Code Submitted

3 Part 3

The model is trained on `train.dat` and tested on `test.dat` with the different set of hyperparameters and their accuracy and training time is recorded.

3.1 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` , `penalty = L2` and `dual = True` for **LinearSVC**.

Table 1: Loss Hyperparameters

Loss	Training Time (in seconds)	Accuracy
Hinge	6.99	0.9888
Squared Hinge	7.58	0.9915

3.2 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 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 :

Table 2: C Hyperparameters in LinearSVC

C value	Training Time (in seconds)	Accuracy
0.01	2.21	0.9865
0.1	2.26	0.9899
1	2.55	0.9918
10	2.49	0.9934
100	2.96	0.9919

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

Table 3: C Hyperparameters in Logistic Regression

C value	Training Time (in seconds)	Accuracy
0.01	0.88	0.9635
0.1	1.16	0.9871
1	1.25	0.9907
10	1.49	0.9922
100	1.99	0.9931

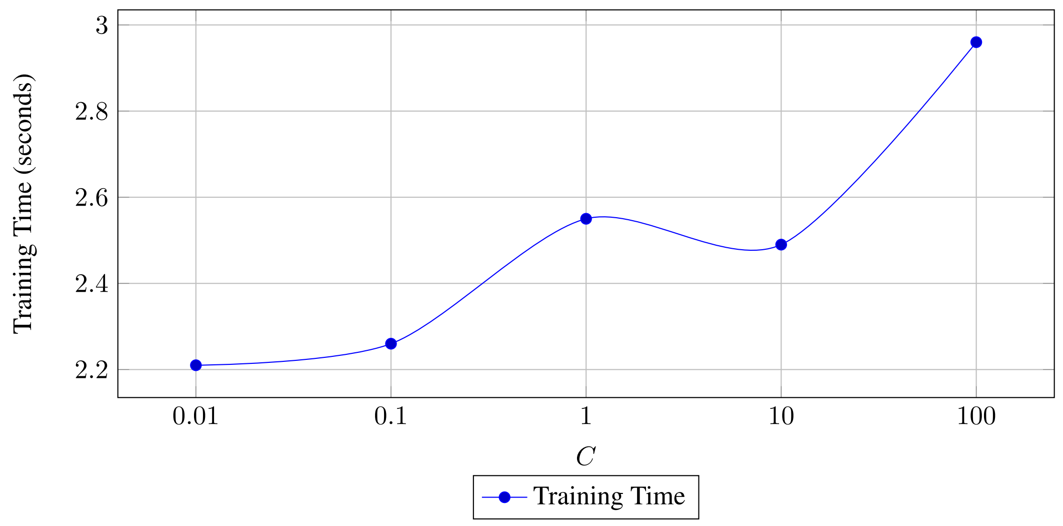

Figure 2: Training Time vs C for LinearSVC

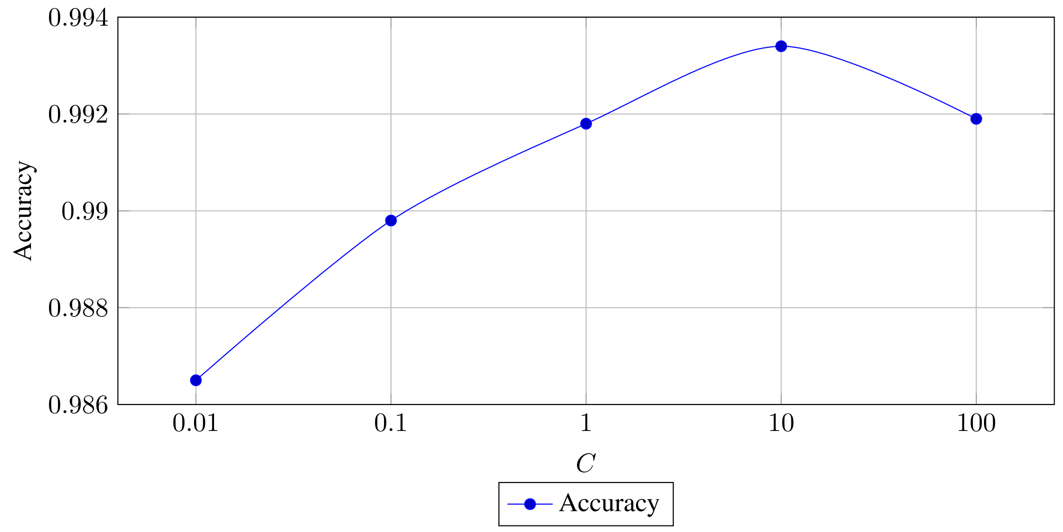

Figure 3: Accuracy vs C for LinearSVC

3.3 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 `C=10`, `loss="squared_hinge"`, `max_iter=1000`, `penalty='l2'`, `dual=False` in **LinearSVC**.

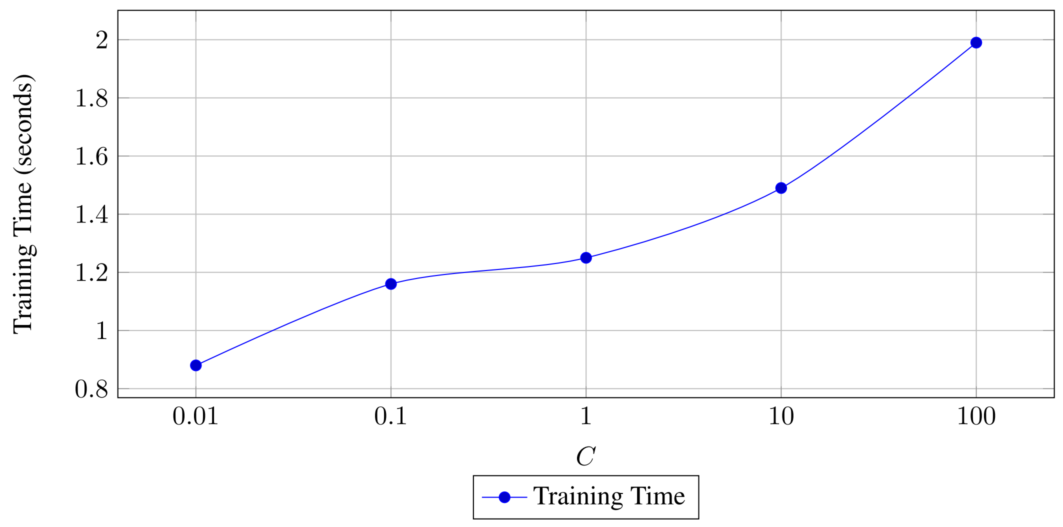

Figure 4: Training Time vs C for Logistic Regression

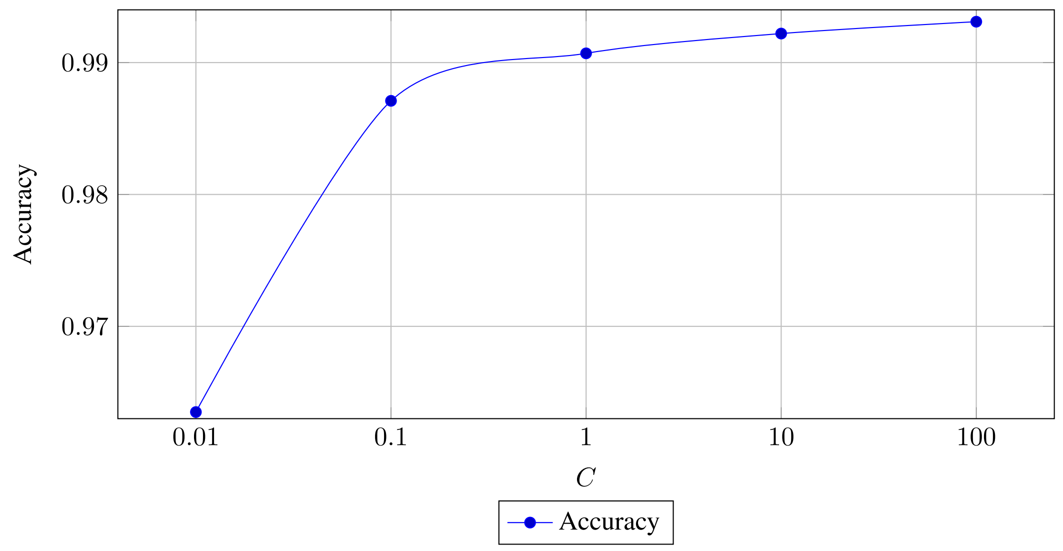

Figure 5: Accuracy vs C for Logistic Regression

Table 4: tol Hyperparameters in LinearSVC

tol value	Training Time	Accuracy
10^{-1}	2.82s	0.9865
10^{-2}	2.92s	0.9899
10^{-3}	6.69s	0.9934
10^{-4}	8.60s	0.9929
10^{-5}	7.39s	0.9930

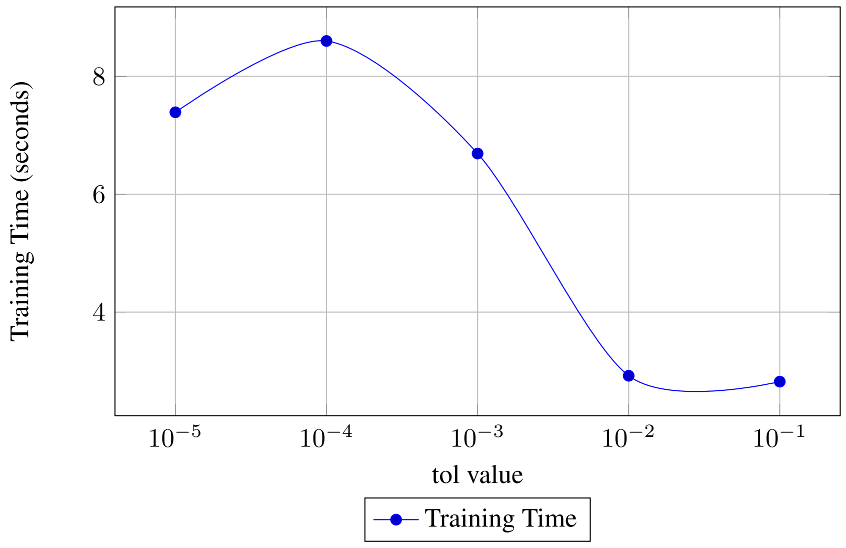

Figure 6: Training Time vs tol value

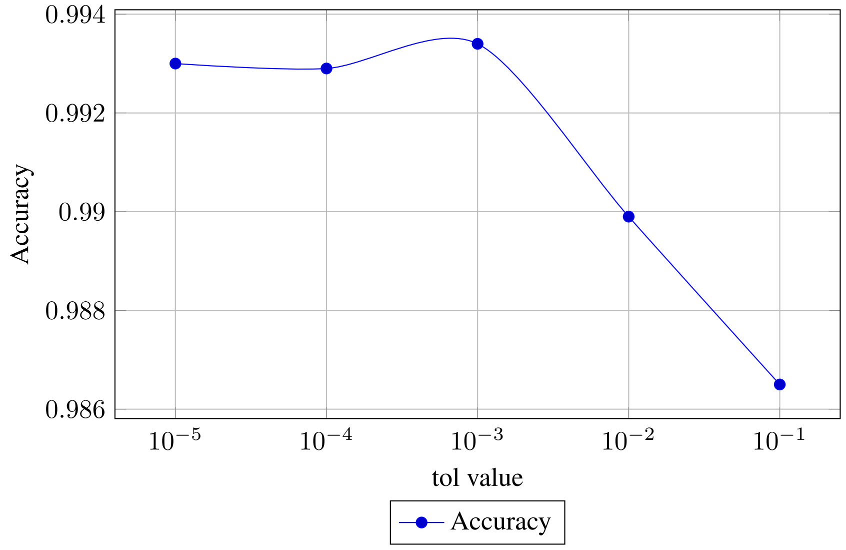

Figure 7: Accuracy vs tol value

We made the observations that are mentioned below using the hyperparameters `C=100`, `solver='lbfgs'`, `max_iter=1000`, `penalty='l2'` in **LogisticRegression**.

Table 5: tol Hyperparameters in Logistic Regression

tol value	Training Time	Accuracy
10^{-1}	1.36s	0.9931
10^{-2}	2.35s	0.9931
10^{-3}	3.98s	0.9931
10^{-4}	3.64s	0.9931
10^{-5}	3.80s	0.9931

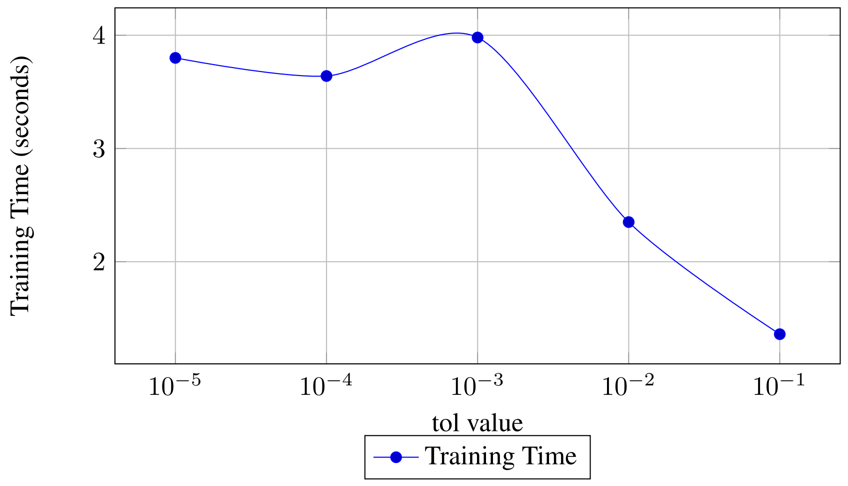

Figure 8: Training Time vs tol value

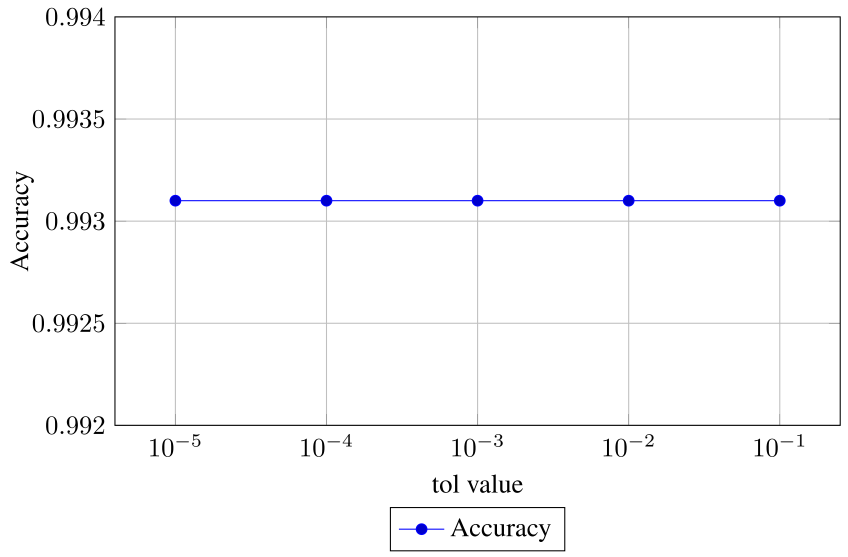

Figure 9: Accuracy vs tol value

3.4 Effect of changing the Penalty hyperparameter in LinearSVC and Logistic Regression

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.

`asC=1`, `tolerance = 1e-3`, `loss = squared hinge` and `dual = False` **LinearSVC**.

Table 6: Penalty Hyperparameters in LinearSVC

Penalty	Training Time (in seconds)	Accuracy
L1	86.04s	0.9915
L2	2.55s	0.9918

Table 7: Penalty Hyperparameters in Logistic Regression

Penalty	Training Time (in seconds)	Accuracy
L1	55.75	0.9915
L2	1.20	0.9907

The **liblinear** solver was used with the L1 penalty. For the L2 penalty, **lbfgs**, the default solver was used.
Since in L1 penalty does not allow using **lbfgs** solver.