article

CS771 : Introduction to Machine Learning
Assignment - 1
Team: Syntax Killer

Dhruv Varshney
220366
Dept. of Electrical Engineering
IIT Kanpur
vdhruv22@iitk.ac.in

Sarthak Paswan
220976
Dept. of Computer Science and Engineering
IIT Kanpur
sarthakp22@iitk.ac.in

Ansh Jain
220166
Dept. of Civil Engineering
IIT Kanpur
anshjain22@iitk.ac.in

Shyam Sundar
221049
Dept. of Mathematics and Scientific Computing
IIT Kanpur
shyams22@iitk.ac.in

Rudraksh Pal Singh
220920
Dept. of Mechanical Engineering
IIT Kanpur
rudrakshps22@iitk.ac.in

Daniel Siwakoti
231040127
Dept. of Electrical Engineering
IIT Kanpur
daniels23@iitk.ac.in

Abstract

This is our solution to the first Assignment of the course CS771. We are required to show(mathematical derivation) how for a simple arbiter PUF, a linear model can predict the time it takes for the upper signal to reach the finish line. Then using that derivation to derive how a linear model can predict Response0 and Response1 for a COCO-PUF. Fianlly, we have to report the outcomes of experiments with both the sklearn.svm.LinearSVC and sklearn.linear model.LogisticRegression methods when used to learn the linear model for the COCO-PUF, how various hyperparameters affected training time and test accuracy.

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 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 find the time taken by upper signal to reach the finish line

It has been found that knowing time taken by a relatively small 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:

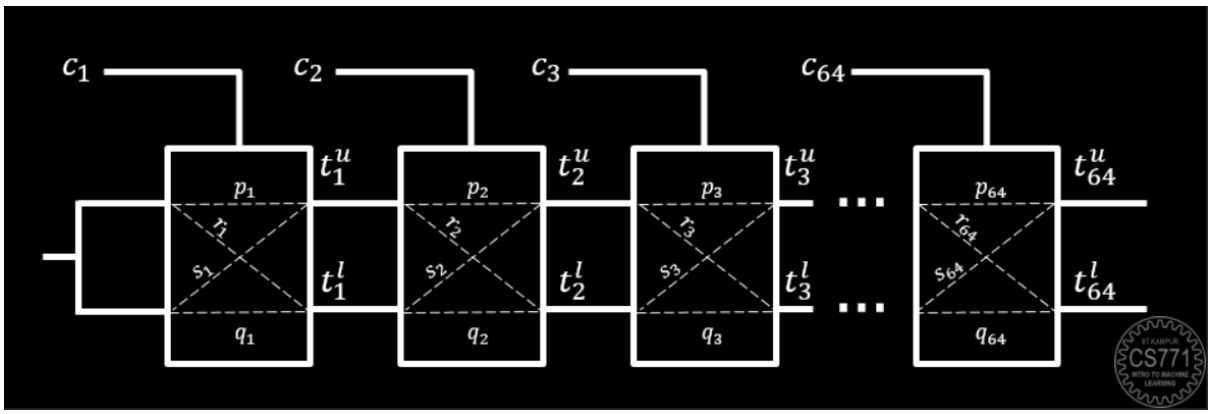

Figure 1: A simple arbiter PUF with 64 multiplexers

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

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

Now, as given in the lecture slides -

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

Adding equations (1) and (2), we get -

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

For n = 0,

$$t_0^u + t_0^l = (1 - c_0)(p_0 + q_0) + c_0(r_0 + s_0) \quad \text{(Since, } t_{-1}^u = 0 \text{ \& } t_{-1}^l = 0)$$

For n = 1,

$$t_1^u + t_1^l = (1 - c_0).(p_0 + q_0) + c_0.(r_0 + s_0) + (1 - c_1).(p_1 + q_1) + c_1.(r_1 + s_1)$$

For n = 2,

$t_2^u + t_2^l = (1 - c_0).(p_0 + q_0) + c_0.(r_0 + s_0) + (1 - c_1).(p_1 + q_1) + c_1.(r_1 + s_1) + (1 - c_2).(p_2 + q_2) + c_2.(r_2 + s_2)$

By induction-

$$t_{31}^u + t_{31}^l = \sum_{i=0}^{31}[(1 - c_i).(p_i + q_i) + c_i.(r_i + s_i)] \tag{3}$$

From the lecture slides we have expression for the difference of the equation (1) and (2),

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

where,

$$\begin{aligned} \Delta_i &= t_i^u - t_i^l \\ d_i &= (1 - 2c_i) \\ \alpha_i &= (p_i - q_i + r_i - s_i)/2 \\ \beta_i &= (p_i - q_i - r_i + s_i)/2 \end{aligned}$$

For i = 0,

$$\Delta_0 = \alpha_0.d_0 + \beta_0 \quad (\text{Since, } \Delta_{-1} = 0)$$

Let us take -

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

Then,

$$\Delta_0 = A_0$$

For i = 1,

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

$$\Delta_1 = d_1.A_0 + A_1$$

For i = 2,

$$\Delta_2 = d_2.d_1.A_0 + d_2.A_1 + A_2$$

Similarly,

$$\Delta_{31} = \sum_{i=0}^{31} A_i. \prod_{j=i+1}^{31} d_j$$

So,

$$t_{31}^u - t_{31}^l = \sum_{i=0}^{31}[(p_i - q_i)(1 - c_i) + c_i.(s_i - r_i)] \prod_{j=i+1}^{31} (1 - 2c_j) \tag{4}$$

$$\tag{5}$$

Adding equation (3) and (4), we get,

$$t^u_{31} = t^u = \frac{1}{2}.\sum_{i=0}^{31}(1-c_i)(p_i+q_i) + c_i.(r_i+s_i) + \frac{1}{2}.\sum_{i=0}^{31}[(1-c_i).(p_i-q_i) + c_i.(s_i-r_i)] \prod_{j=i+1}^{31}(1-2c_j) \tag{6}$$

If we take,

$$\begin{aligned} d_i &= (1-2.c_i) \\ \alpha_i &= \frac{(p_i+q_i+r_i+s_i)}{4} \\ \beta_i &= \frac{(p_i+q_i-r_i-s_i)}{4} \\ \gamma_i &= \frac{(p_i-q_i+s_i-r_i)}{4} \\ \delta_i &= \frac{(p_i-q_i+r_i-s_i)}{4} \end{aligned}$$

And simplyfying equation (5), we get -

$$t^u = \sum_{i=0}^{31}\alpha_i + \gamma_{31} + \sum_{i=0}^{31} d_i.\beta_i + \sum_{i=0}^{31}(\delta_i + \gamma_{i-1})(d_i.d_{i+1}.d_{i+2}.....d_{31}) \quad \text{(Here, } \gamma_{-1} = 0\text{)}$$

$$\text{Take b} = \sum_{i=0}^{31}\alpha_i + \gamma_{31}$$

Now, converting this to matrix form, we get -

$$t^u(c) = W^T\phi(c) + b$$

where,

$$W = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_0 \\ (\delta_1+\gamma_0) \\ \vdots \\ (\delta_{31}+\gamma_{30}+\beta_{31}) \end{pmatrix}$$

and,

$$\phi(c) = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{30} \\ d_0d_1d_2\cdots d_{31} \\ d_1d_2d_3\cdots d_{31} \\ \vdots \\ d_{31} \end{pmatrix}$$

2 Dimensionality of the model W

From previous part,

$$W = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_0 \\ (\delta_1 + \gamma_0) \\ \vdots \\ (\delta_{31} + \gamma_{30} + \beta_{31}) \end{pmatrix}$$

Clearly, it is **63 x 1** matrix.
Hence, dimension of the **W**(model) is **63 x 1**.

3 Linear models to predict Response0 and Response1

1.3 Another type of PUF — COCO-PUF

A COCO-PUF uses 2 arbiter PUFs, say PUF0 and PUF1 – each PUF has its own set of multiplexers with possibly different delays. Given a challenge, it is fed into both the PUFs. Instead of the lower and upper signals of PUF0 competing with each other, the lower signal from PUF0 compete with the lower signal from PUF1 using an arbiter called Arbiter0 to generate a response called Response0. Also, the upper signal from PUF0 compete with the upper signal from PUF1 using a second arbiter called Arbiter1 to generate a response called Response1.

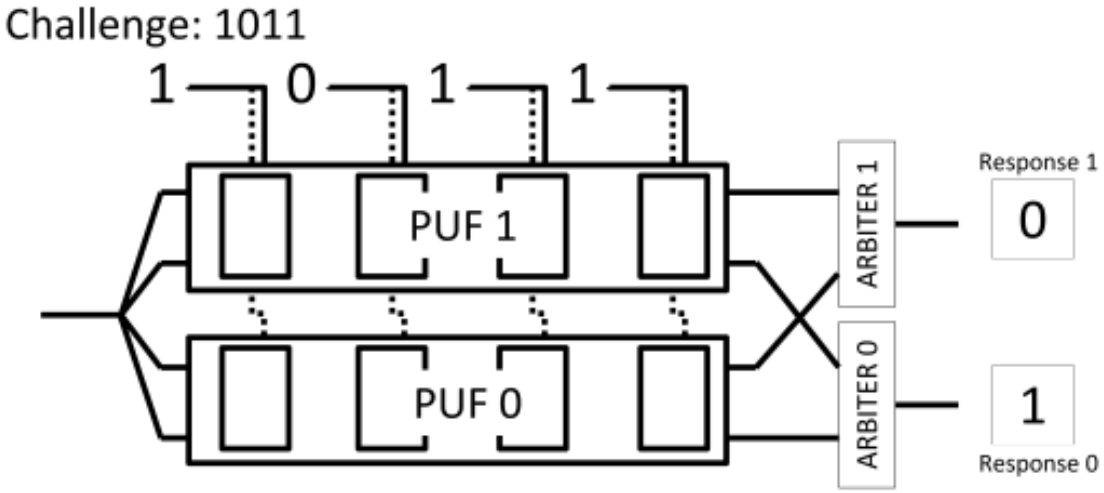

Figure 2: A COCO-PUF with 4-bit challenges and 2-bit responses

1.4 Using ML to crack COCO-PUF

From the previous part, the expression for t_{31}^u is:

$$t_{31}^u = \frac{1}{2}.\sum_{i=0}^{31}(1-c_i)(p_i+q_i) + c_i.(r_i+s_i) + \frac{1}{2}.\sum_{i=0}^{31}[(1-c_i).(p_i-q_i) + c_i.(s_i-r_i)] \prod_{j=i+1}^{31}(1-2c_j)$$

Similar expression for t_{31}^{l} can be derived by subtracting equation (4) from (3),

$$t_{31}^{l} = \frac{1}{2}.\sum_{i=0}^{31}(1-c_i)(p_i+q_i) + c_i.(r_i+s_i) - \frac{1}{2}.\sum_{i=0}^{31}[(1-c_i).(p_i-q_i) + c_i.(s_i-r_i)] \prod_{j=i+1}^{31}(1-2c_j)$$

$$2t_{31}^{l_1} = \sum_{i=0}^{31}(1-c_i)(p_i^1+q_i^1) + c_i(s_i^1+r_i^1) - \sum_{i=0}^{31}\left((1-c_i)(p_i^1-q_i^1) + c_i(s_i^1-r_i^1)\right)\left(\prod_{j=i+1}^{31}(1-2c_j)\right) .....(7)$$

$$2t_{31}^{l_0} = \sum_{i=0}^{31}(1-c_i)(p_i^0+q_i^0) + c_i(s_i^0+r_i^0) - \sum_{i=0}^{31}\left((1-c_i)(p_i^0-q_i^0) + c_i(s_i^0-r_i^0)\right)\left(\prod_{j=i+1}^{31}(1-2c_j)\right) .....(8)$$

Subtracting equation (8) from (7),

$$\begin{aligned}2(t_{31}^{l_0} - t_{31}^{l_1}) = &\sum_{i=0}^{31}(1-c_i)\left(p_i^0+q_i^0-p_i^1-q_i^1\right) + c_i\left(s_i^0+r_i^0-s_i^1-r_i^1\right)\\ &+\sum_{i=0}^{31}\left((1-c_i)\left(p_i^1-q_i^1-p_i^0+q_i^0\right) + c_i\left(s_i^1-r_i^1-s_i^0+r_i^0\right)\right)\left(\prod_{j=i+1}^{31}(1-2c_j)\right)\end{aligned}$$

$$let, (1-2c_i) = d_i$$

$$\begin{aligned}2(t_{31}^{l_0} - t_{31}^{l_1}) = &\sum_{i=0}^{31}\left(\frac{1}{2}+\frac{d_i}{2}\right)\left(p_i^0+q_i^0-p_i^1-q_i^1\right) + \left(\frac{1-d_i}{2}\right)\left(s_i^0+r_i^0-s_i^1-r_i^1\right)\\ &+\sum_{i=0}^{31}\left(\frac{1}{2}+\frac{d_i}{2}\right)(d_{i+1}...d_{31})\left(p_i^1-q_i^1-p_i^0+q_i^0\right) + \left(\frac{1-d_i}{2}\right)(d_{i+1}...d_{31})\left(s_i^1-r_i^1-s_i^0+r_i^0\right)\end{aligned}$$

$$\begin{aligned}t_{31}^{l_0} - t_{31}^{l_1} = &\sum_{i=0}^{31}\left(\frac{p_i^0+q_i^0+s_i^0+r_i^0-p_i^1-q_i^1-s_i^1-r_i^1}{4}\right) + d_i\left(\frac{p_i^0+q_i^0-s_i^0-r_i^0-p_i^1-q_i^1+s_i^1+r_i^1}{4}\right)\\ &+(d_{i+1}\dots d_{31})\left(\frac{p_i^1-q_i^1-p_i^0+q_i^0+s_i^1-r_i^1-s_i^0+r_i^0}{4}\right)\\ &+(d_i\dots d_{31})\left(\frac{p_i^1-q_i^1-p_i^0+q_i^0-s_i^1+r_i^1+s_i^0-r_i^0}{4}\right)\end{aligned}$$

Let,

$$\begin{aligned}\alpha_i &= \frac{p_i^0+q_i^0+s_i^0+r_i^0-p_i^1-q_i^1-s_i^1-r_i^1}{4}\\ \beta_i &= \frac{p_i^0+q_i^0-s_i^0-r_i^0-p_i^1-q_i^1+s_i^1+r_i^1}{4}\\ \gamma_i &= \frac{p_i^1-q_i^1-p_i^0+q_i^0+s_i^1-r_i^1-s_i^0+r_i^0}{4}\\ \delta_i &= \frac{p_i^1-q_i^1-p_i^0+q_i^0-s_i^1+r_i^1+s_i^0-r_i^0}{4}\end{aligned}$$

$$t_{31}^{l_0} - t_{31}^{l_1} = \sum_{i=0}^{31}\alpha_i + \sum_{i=0}^{31}d_i\beta_i + \sum_{i=0}^{31}(d_{i+1}\dots d_{31})\gamma_i + \sum_{i=0}^{31}(d_i\dots d_{31})\delta_i$$

$$t_{31}^{l_0} - t_{31}^{l_1} = \left(\sum_{i=0}^{31} \alpha_i + \gamma_{31}\right) + \left(\sum_{i=0}^{30} d_i\beta_i\right) + \delta_1(d_1....d_{31}) + (\delta_2+\gamma_1)(d_2...d_{31}) + ... + (\delta_{31}+\gamma_{30}+\beta_{31})d_{31}$$

$$Let, \widetilde{b} = \sum_{i=0}^{31} \alpha_i + \gamma_{31}$$

$$t_{31}^{l_0} - t_{31}^{l_1} = \widetilde{b} + [\beta_0 \quad \beta_1 \quad \dots \quad \beta_{30} \quad \delta_1 \quad (\delta_2+\gamma_1) \quad \dots \quad (\delta_{31}+\gamma_{30}+\beta_{31})] \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{30} \\ d_0d_1....d_{31} \\ d_1d_2....d_{31} \\ \vdots \\ d_{30}d_{31} \\ d_{31} \end{bmatrix}$$

$$t_{31}^{l_0} - t_{31}^{l_1} = \widetilde{b} + \widetilde{W_0}^T \widetilde{\phi_0(C)} \quad \text{, where}$$

$$\widetilde{W_0} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_1 \\ \delta_2+\gamma_1 \\ \vdots \\ \delta_{31}+\gamma_{30}+\beta_{31} \end{bmatrix} \quad \text{and} \quad \widetilde{\phi_0(C)} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{30} \\ d_0d_1 \dots d_{31} \\ d_1d_2 \dots d_{31} \\ \vdots \\ d_{31} \end{bmatrix}$$

Hence the Response0 can predicted using this linear model as -

$$\frac{1+\text{sign}(\widetilde{\mathbf{W_0}}^\top \widetilde{\phi_0}(\mathbf{c}) + \widetilde{\mathbf{b}})}{2} = r^0(\mathbf{c})$$

Similar to the Response0, we can predict Respone1 by using the expression of t_{31}^{u0} and t_{31}^{u1} in equation (7) and (8) respectively, we get-

$$t_{31}^{u_0} - t_{31}^{u_1} = \sum_{i=0}^{31} \alpha_i + \sum_{i=0}^{31} d_i\beta_i + \sum_{i=0}^{31} (d_{i+1} \dots d_{31})\gamma_i + \sum_{i=0}^{31} (d_i \dots d_{31})\delta_i$$

Here,

$$\alpha_i = \frac{p_i^0 + q_i^0 + s_i^0 + r_i^0 - p_i^1 - q_i^1 - s_i^1 - r_i^1}{4}$$
$$\beta_i = \frac{p_i^0 + q_i^0 - s_i^0 - r_i^0 - p_i^1 - q_i^1 + s_i^1 + r_i^1}{4}$$
$$\gamma_i = \frac{p_i^0 - q_i^0 - p_i^1 + q_i^1 + s_i^0 - r_i^0 - s_i^1 + r_i^1}{4}$$
$$\delta_i = \frac{p_i^0 - q_i^0 - p_i^1 + q_i^1 - s_i^0 + r_i^0 + s_i^1 - r_i^1}{4}$$

Note: Here the value for the γ_i and δ_i in Response0 and Response1 are slighty different.
So finally the Resposne1 can similarly be predicted as -

$$\frac{1+\text{sign}(\widetilde{\mathbf{W_1}}^\top \widetilde{\phi_1}(\mathbf{c}) + \widetilde{\mathbf{b}})}{2} = r^1(\mathbf{c})$$

Here,

$$\widetilde{W_1} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_1 \\ \delta_2 + \gamma_1 \\ \vdots \\ \delta_{31} + \gamma_{30} + \beta_{31} \end{bmatrix} \quad \text{and} \quad \widetilde{\phi_1(C)} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{30} \\ d_0 d_1 \dots d_{31} \\ d_1 d_2 \dots d_{31} \\ \vdots \\ d_{31} \end{bmatrix}$$

4 Dimensionality of the linear models $\mathbf{W_0}$ and $\mathbf{W_1}$

From previous part -

$$\widetilde{W_0} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_1 \\ \delta_2 + \gamma_1 \\ \vdots \\ \delta_{31} + \gamma_{30} + \beta_{31} \end{bmatrix} \quad \text{and} \quad \widetilde{W_1} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{30} \\ \delta_1 \\ \delta_2 + \gamma_1 \\ \vdots \\ \delta_{31} + \gamma_{30} + \beta_{31} \end{bmatrix}$$

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 **63 x 1** matrices.
Hence, the dimensionality for both the $\widetilde{W_0}$ and $\widetilde{W_1}$ is **63 x 1**.

5 Code for the two linear models $\mathbf{W_0}$, $\mathbf{b_0}$, $\mathbf{W_1}$, $\mathbf{b_1}$

Code Submitted

6 Outcomes of the Experiments

6.1 Training Set

The model is trained on `public_trn.txt` and tested on `public_tst.txt` with the different set of hyperparameters and their accuracy and training time is recorded.

a) 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 y0**	**Accuracy y1**
Hinge	1.87	0.9841	0.9961
Squared Hinge	5.66	0.9800	0.9979

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-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

C value	**Training Time (in seconds)**	**Accuracy y0**	**Accuracy y1**
0.01	1.43	0.9799	0.9948
0.1	1.41	0.9803	0.9964
1	1.51	0.9800	0.9980
10	1.57	0.9800	0.9987
100	1.52	0.9800	0.9989

Table 3: C Hyperparameters in Logistic Regression

C value	**Training Time (in seconds)**	**Accuracy y0**	**Accuracy y1**
0.01	0.90	0.9800	0.9931
0.1	1.02	0.9805	0.9956
1	1.01	0.9806	0.9970
10	1.03	0.9808	0.9981
100	1.02	0.9808	0.9990

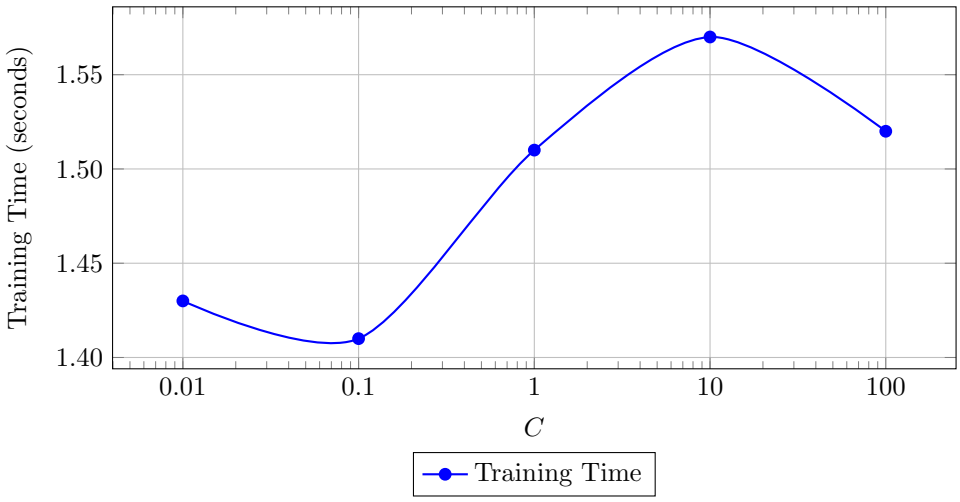

Figure 3: Training Time vs C for LinearSVC

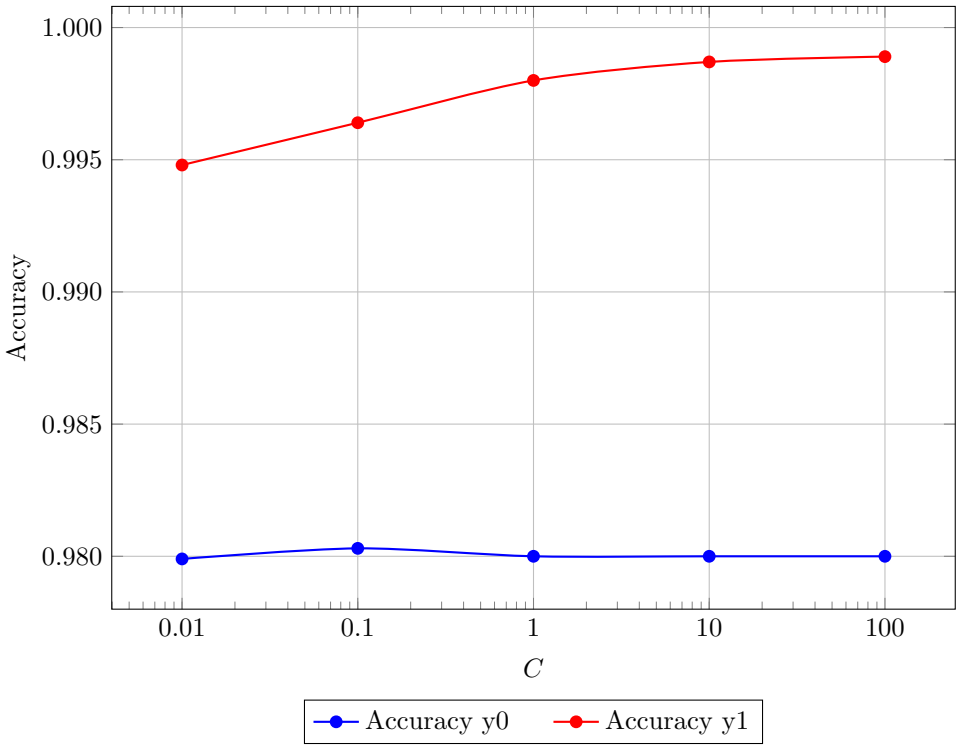

Figure 4: Accuracy vs C for LinearSVC

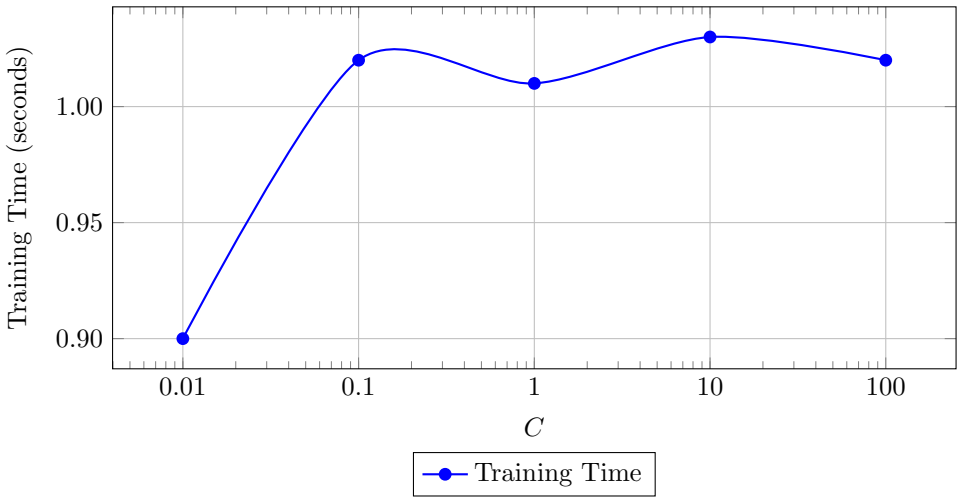

Figure 5: Training Time vs C for Logistic Regression

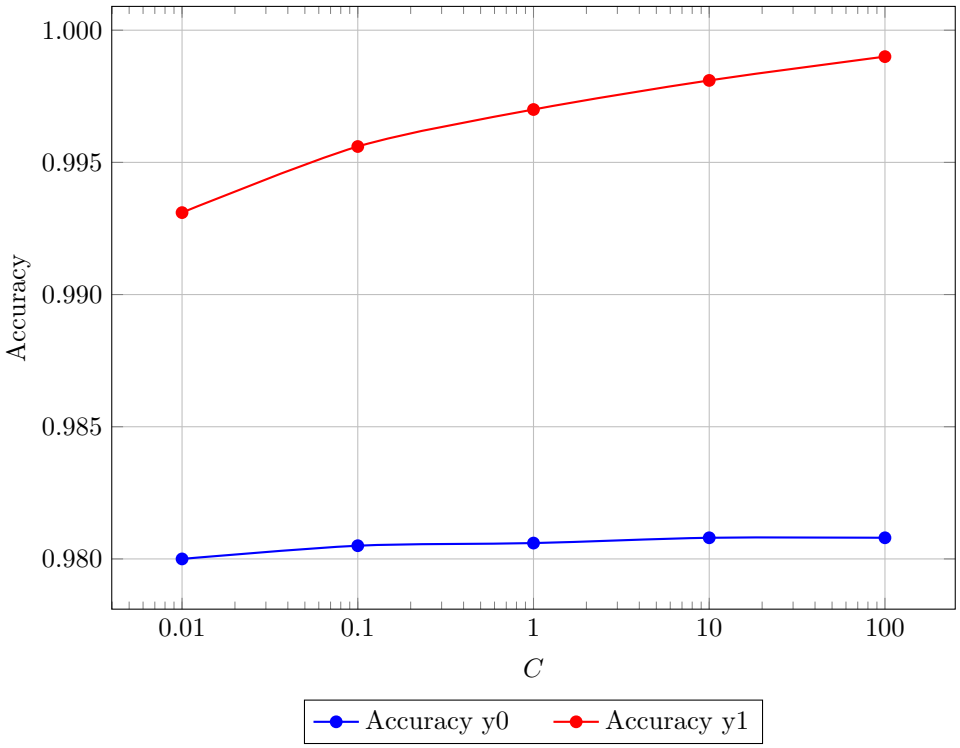

Figure 6: Accuracy vs C for Logistic Regression

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=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 seconds)	Accuracy y0	Accuracy y1
10^{-1}	1.28	0.9800	0.9923
10^{-2}	1.45	0.9800	0.9961
10^{-3}	1.34	0.9800	0.9987
10^{-4}	1.49	0.9800	0.9987
10^{-5}	1.51	0.9800	0.9989

Table 5: tol Hyperparameters in Logistic Regression

C value	Training Time (in seconds)	Accuracy y0	Accuracy y1
10^{-1}	1.39	0.9808	0.9990
10^{-2}	1.20	0.9808	0.9990
10^{-3}	1.18	0.9808	0.9990
10^{-4}	0.99	0.9808	0.9990
10^{-5}	1.01	0.9808	0.9990

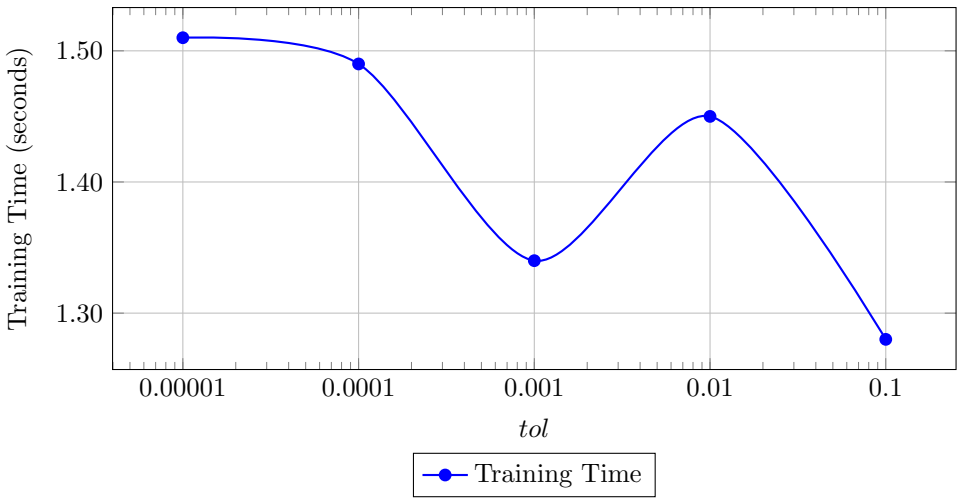

Figure 7: Training Time vs *tol* for LinearSVC

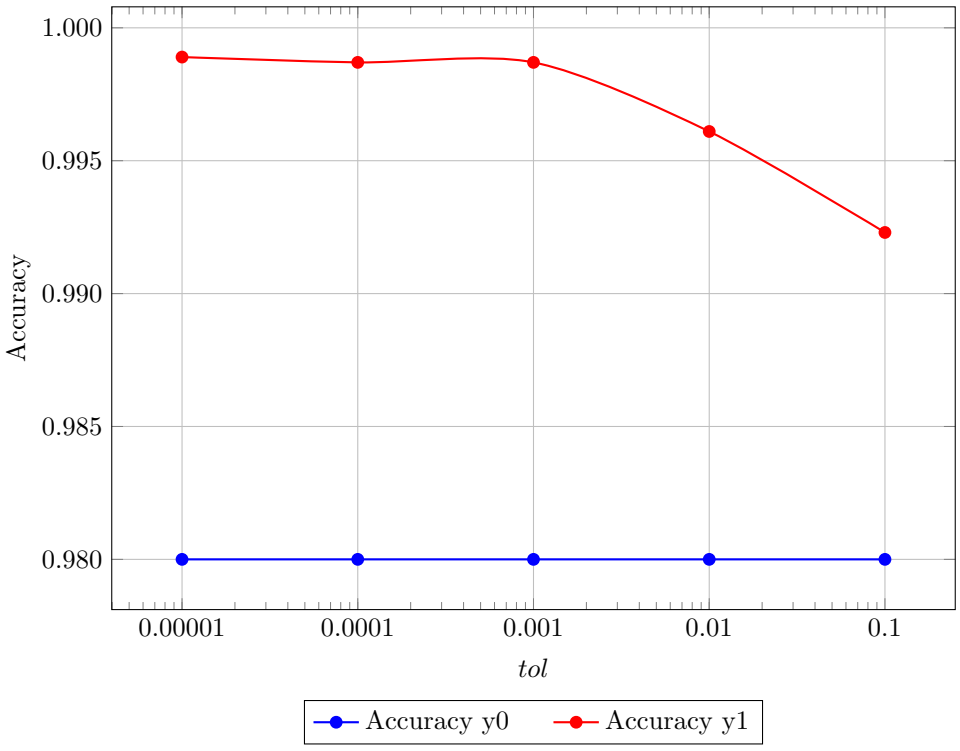

Figure 8: Accuracy vs *tol* for LinearSVC

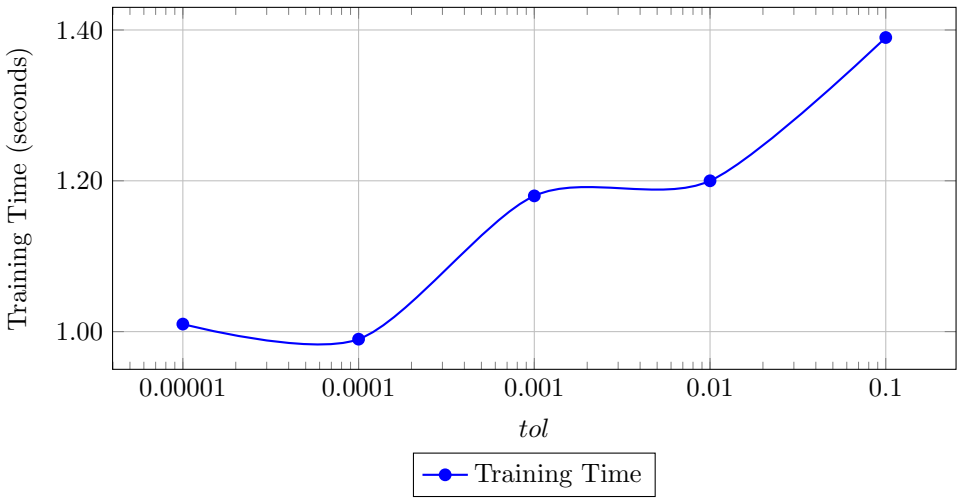

Figure 9: Training Time vs *tol* for Logistic Regression

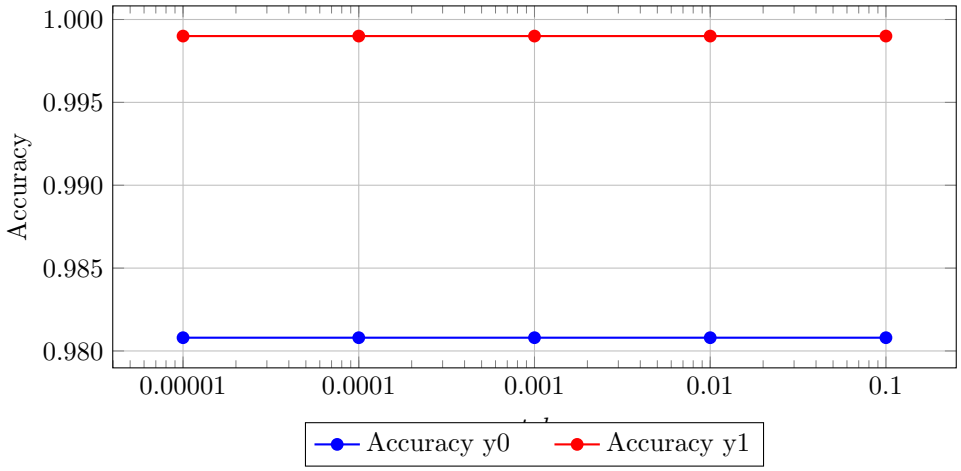

Figure 10: Accuracy vs *tol* for Logistic Regression

d) Effect of changing the *penlaty* 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 `loss = Squared Hinge`, `C=1`, `tolerance = 1e-3` and `dual = False` for **LinearSVC** and `C=100`, `tolerance = 1e-3`, `dual = False` for **Logistic Regression**:

Table 6: Penalty Hyperparameters in LinearSVC

Penalty	**Training Time (in seconds)**	**Accuracy y0**	**Accuracy y1**
L1	17.68	0.9801	0.9979
L2	1.53	0.9800	0.9980

Table 7: Penalty Hyperparameters in Logistic Regression

Solver	**Penalty**	**Training Time (in seconds)**	**Accuracy y0**	**Accuracy y1**
liblinear	L1	18.12	0.9808	0.9982
lbfgs	L2	1.25	0.9808	0.9990

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.