CS771 Assignment-1

Melbo's Team
Garvit Arora(200372)
Nikhil Gupta (200634)
Shivam Singhal (200939)
Shrey Wadhawan (200948)

1 Detailed mathematical derivation of a simple XORRO PUF

Consider a *XORRO* consisting of *R* XORs, where i_{th} XOR is receiving input p_i from the previous XOR in the present cycle and a_i as the challenge bit. Then,

$$\delta^i_{p_i} = \delta^i_{00}(1-a_i)(1-p_i) + \delta^i_{01}(1-a_i)(p_i) + \delta^i_{10}(a_i)(1-p_i) + \delta^i_{11}(a_i)(p_i)$$

Here, $p_i \in \{0,1\}$ as per whatever bit is received in the i^{th} XOR from the $(i-1)^{th}$ XOR.
Computing δ^i for one complete cycle:

$$\begin{aligned}
\delta^i &= \sum_{p_i \in \{0,1\}} \delta^i_{p_i} \\
\delta^i &= \delta^i_{00}(1-a_i) + \delta^i_{01}(1-a_i) + \delta^i_{10}(a_i) + \delta^i_{11}(a_i) \\
\delta^i &= a_i[-\delta^i_{00} - \delta^i_{01} + \delta^i_{10} + \delta^i_{11}] + [\delta^i_{00} + \delta^i_{01}] \\
\delta^i &= a_i\alpha_i + \beta_i
\end{aligned}$$

For a XORRO, consisting of R XORs, the total delay can be expressed as:

$$\begin{aligned}
\delta &= \sum_{i=1}^{R} \delta^i = \sum_{i=1}^{R}(a_i\alpha_i + \beta_i) = \sum_{i=1}^{R} a_i\alpha_i + \sum_{i=1}^{R}\beta_i \\
\delta &= \sum_{i=1}^{R}(a_i\alpha_i) + \beta
\end{aligned}$$

For two XORROs the difference in the delays, as computed from the previous equation, is used to calculate the output of the simple XORRO PUF.
We define Δ as:

$$\begin{aligned}
\Delta &= \delta_{lower} - \delta_{upper} \\
&= \sum_{i=1}^{R}[(\alpha_i a_i)_{lower} - (\alpha_i a_i)_{upper}] + (\beta_{lower} - \beta_{upper}) \\
&= \sum_{i=1}^{R}[\alpha_{i_{lower}} - \alpha_{i_{upper}}] \cdot a_i + b \\
&= \mathbf{w}^T\phi(\mathbf{c}) + b
\end{aligned}$$

Preprint. Under review.

Here,

$$\mathbf{w} = \begin{bmatrix} \alpha_{1_{upper}} \\ \alpha_{2_{upper}} \\ \vdots \\ \alpha_{R_{upper}} \\ \alpha_{1_{lower}} \\ \alpha_{2_{lower}} \\ \vdots \\ \alpha_{R_{lower}} \end{bmatrix}_{(2R\times 1)} \quad , \quad \boldsymbol{c} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_R \end{bmatrix}_{(R\times 1)} \quad , \quad \phi(\boldsymbol{c}) = \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{c} \end{bmatrix} \quad = \quad \begin{bmatrix} -a_1 \\ -a_2 \\ \vdots \\ -a_R \\ a_1 \\ a_2 \\ \vdots \\ a_R \end{bmatrix}_{(2R\times 1)}$$

The PUF will output $\mathbf{1}$ if the upper XORRO has a higher frequency as compared to the lower one. This implies that the output of the XORRO PUF will be $\mathbf{1}$ if $\Delta > \mathbf{0}$ and $\mathbf{0}$ otherwise.
Thus, the output of the simple XORRO PUF can be given as:

$$\frac{1 + \texttt{sign}(\Delta)}{2}$$
$$\frac{1 + \texttt{sign}(\mathbf{w}^T \phi(\boldsymbol{c}) + b)}{2}$$

2 Cracking Advanced XORRO

We employed two approaches to handle Advanced XORRO.

2.1 First Approach using M models

$$M = 2^{s-1}(2^s - 1)$$

Since,

$$s = 4 \Rightarrow M = 120$$

Multiplexer Output:

$$\texttt{Output} = (1 - s_0)(1 - s_1)(1 - s_2)(1 - s_3)X_0 + ... + s_0 s_1 s_2 s_3 X_{15}$$

2.2 Second Approach using 1 model

As in the normal XORRO, we have deduced these values for the linear model.

$$\mathbf{w} = \begin{bmatrix} \alpha_{1_{upper}} \\ \alpha_{2_{upper}} \\ \vdots \\ \alpha_{R_{upper}} \\ \alpha_{1_{lower}} \\ \alpha_{2_{lower}} \\ \vdots \\ \alpha_{R_{lower}} \end{bmatrix}_{(2R\times 1)} \quad , \quad \boldsymbol{c} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_R \end{bmatrix}_{(R\times 1)} \quad , \quad \phi(\boldsymbol{c}) = \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{c} \end{bmatrix} \quad = \quad \begin{bmatrix} -a_1 \\ -a_2 \\ \vdots \\ -a_R \\ a_1 \\ a_2 \\ \vdots \\ a_R \end{bmatrix}_{(2R\times 1)}$$

Now, we propose some changes to this linear model to incorporate all XORROs in one model.

$$\mathbf{w} = \begin{bmatrix} \alpha_{0_1} \\ \alpha_{0_2} \\ \vdots \\ \alpha_{0_R} \\ \beta_0 \\ \alpha_{1_1} \\ \alpha_{1_2} \\ \vdots \\ \alpha_{2^s-1_{R-1}} \\ \alpha_{2^s-1_R} \\ \beta_{2^s-1} \end{bmatrix}_{((R+1)\cdot 2^s \times 1)} \quad , \quad \boldsymbol{c} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_R \\ 1 \end{bmatrix}_{((R+1)\times 1)}$$

$$\phi(\boldsymbol{c}) = \begin{bmatrix} (1-p_0)(1-p_1)(1-p_2)\dots(1-p_{s-1})(-\boldsymbol{c}) \\ p_0(1-p_1)(1-p_2)\dots(1-p_{s-1})(-\boldsymbol{c}) \\ \vdots \\ p_0p_1p_2\dots p_{s-1}(-\boldsymbol{c}) \end{bmatrix} + \begin{bmatrix} (1-q_0)(1-q_1)(1-q_2)\dots(1-q_{s-1})(\boldsymbol{c}) \\ q_0(1-q_1)(1-q_2)\dots(1-q_{s-1})(\boldsymbol{c}) \\ \vdots \\ q_0q_1q_2\dots q_{s-1}(\boldsymbol{c}) \end{bmatrix}$$

This modified linear model incorporates all XORROs of Advanced XORRO as the feature vector (1040×1) have parameters from all 16 XORROs. And according to selector inputs of multiplexers, two entries of $\phi(\boldsymbol{c})$ will be $(\boldsymbol{c}\ \&\ -\boldsymbol{c})$ and other 14 entries will be zero.

For given values of selector bits $(\boldsymbol{p}\ \&\ \boldsymbol{q})$:

$$\phi(\boldsymbol{c}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ -a_1 \\ -a_2 \\ \vdots \\ -a_R \\ -1 \\ 0 \\ \vdots \\ 0 \\ a_1 \\ a_2 \\ \vdots \\ a_R \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{((R+1)\cdot 2^s \times 1)}$$

As we can see from above that, for a given sample, we create a (1040×1) feature vector in which 65 bits will be the same as input+bias corresponding to lower XORRO, and 65 bit will be negative of input+bias corresponding to upper XORRO and other 910 bits will be 0.

Based on the above derivation, we can see that this model will work for cracking the Advanced XORRO because, in this model, even though parameters of all the XORROs are present still inputs corresponding to them are eventually zero except for two selected XORROs.

3 Analysing the Consequences of Changing Parameters

We have employed two types of models from Scikit-Learn:

1. LinearSVC()
2. LogisticRegression()

Further, we studied the variation of *training time*, and *accuracy* for different values of hyperparameters:

1. C (penalty parameter)
2. tolerance and penalty type (L1 / L2)

3.1 LinearSVC

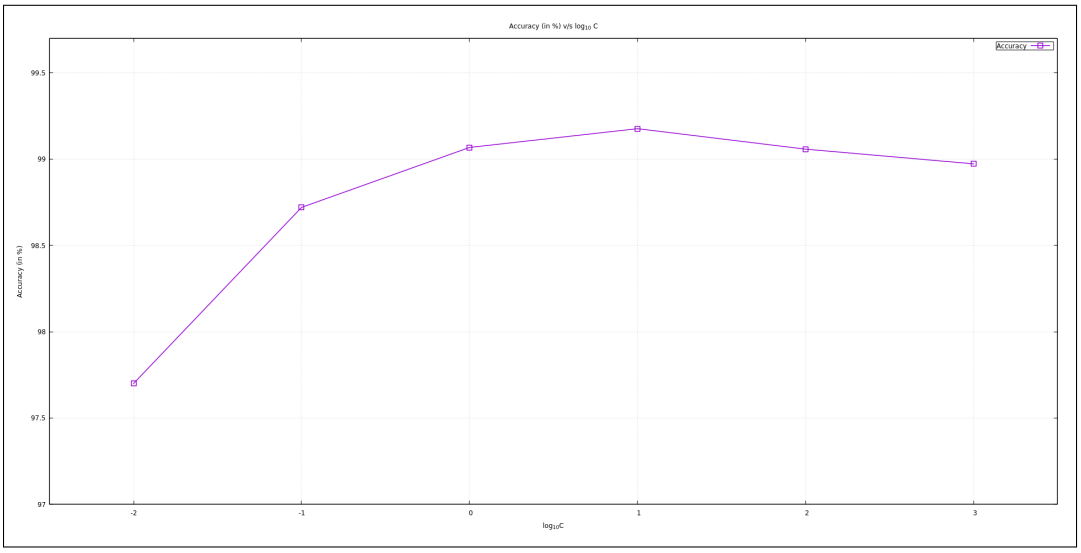

Figure 1: *Accuracy vs C in SVM Model*

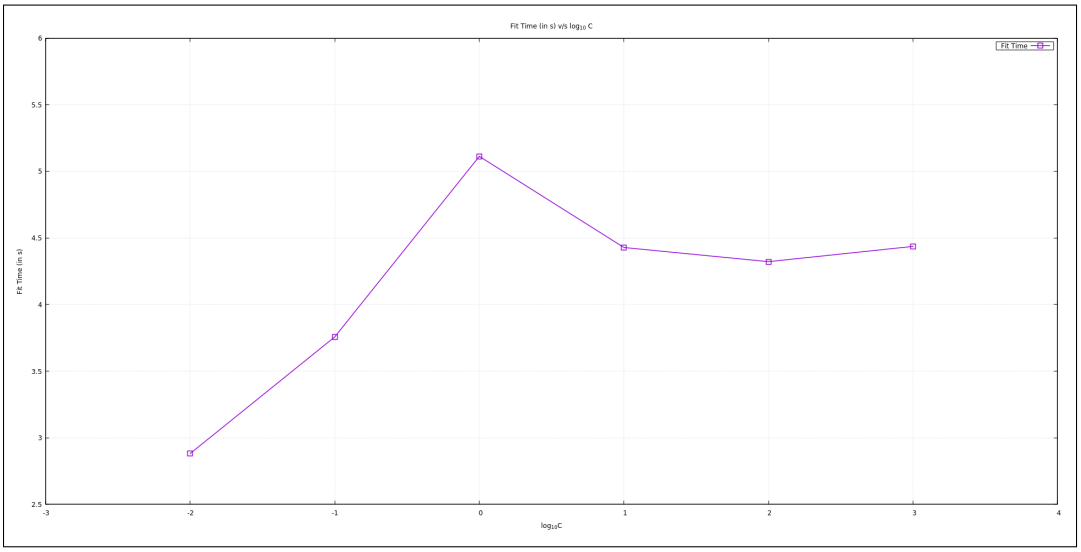

Figure 2: *Fit Time (in s) vs C in SVM Model*

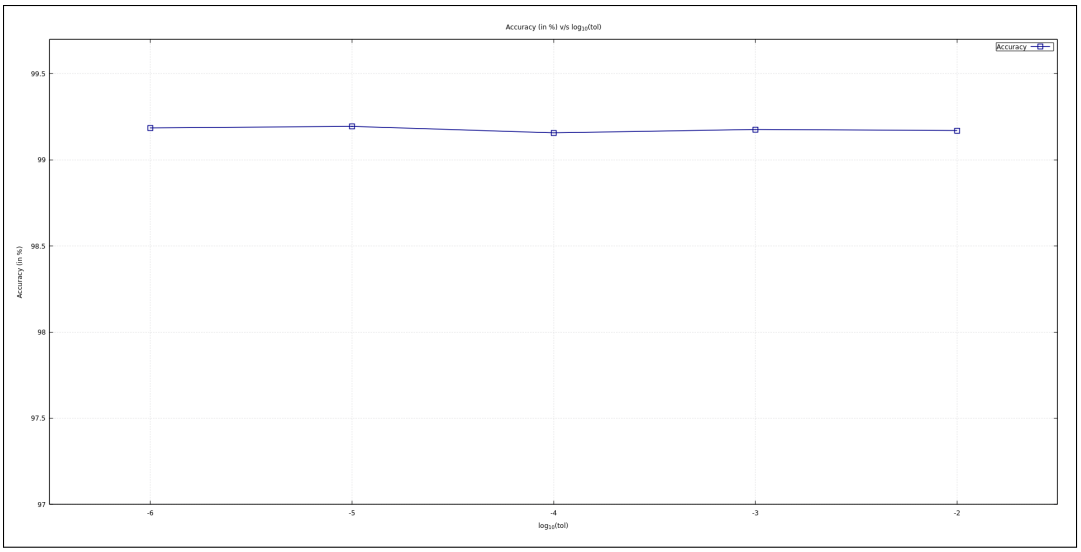

Figure 3: *Accuracy vs tol in SVM Model*

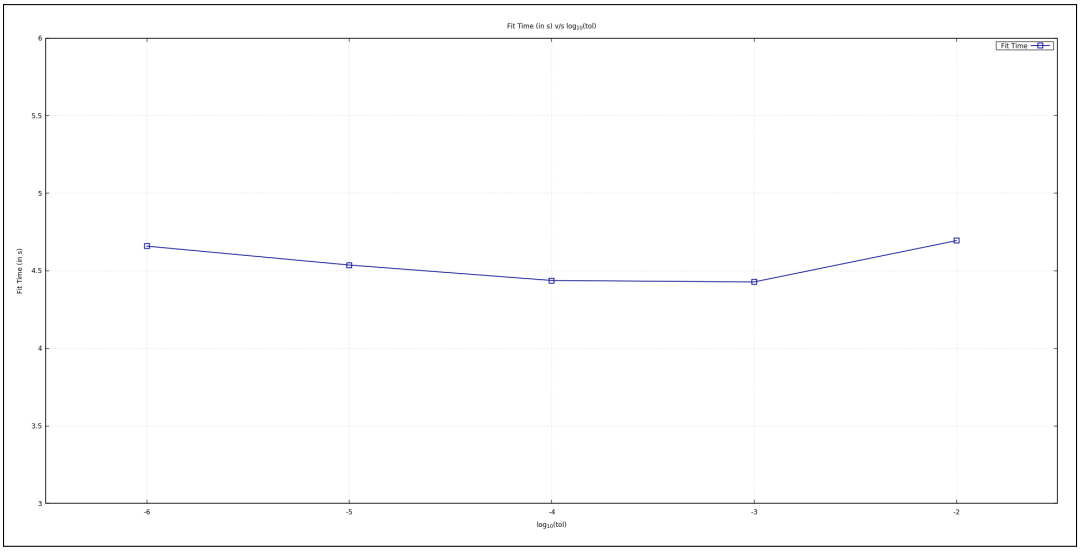

Figure 4: *Fit Time (in s) vs tol in SVM Model*

Fit_Time (in s)	Predict_Time (in s)	Loss	Model_Size	C	Tol	NoIter	Accuracy
4.3589	1.4158	Hinge	8921	50	1e-3	1000	99.0405
4.38921	1.5278	SqHinge	8929	50	1e-3	1000	99.075

Figure 5: *Effect of change of Loss function on Various Parameters in C-SVM Model*

Fit_Time (in s)	Predict_Time (in s)	Loss	Model_Size	C	Tol	NoIter	Accuracy
Penalty L1 (dual = False)							
43.91026	1.4425	SqHinge	8929	10	1e-3	1000	99.2505
Penalty L2 (dual = False)							
3.321174	1.38273	SqHinge	8929	10	1e-3	1000	99.245

Figure 6: *Effect of change of Penalty on Various Parameters in C-SVM Model*

3.2 Logistic Regression:

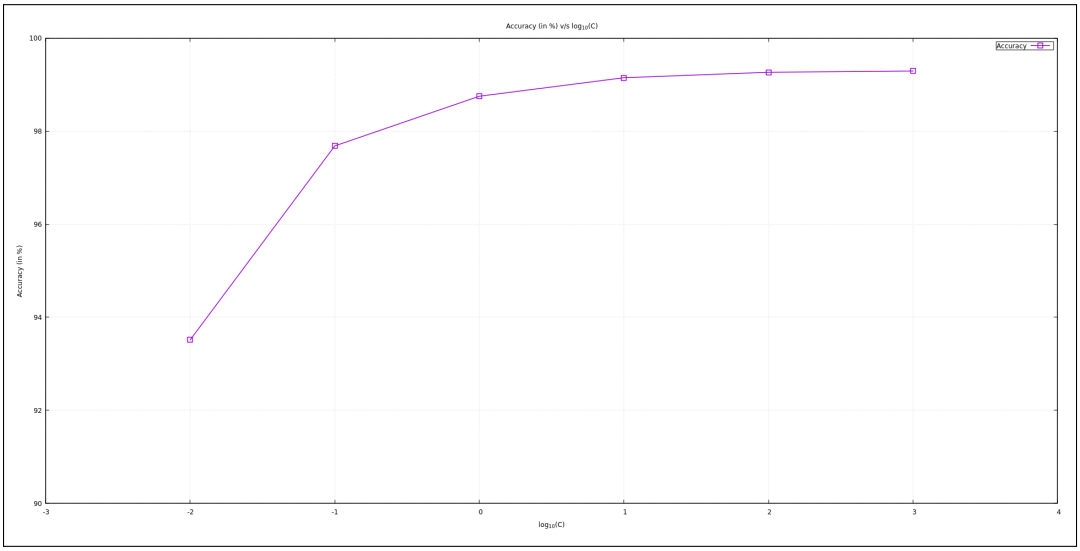

Figure 7: *Accuracy vs C in Logistic Regression*

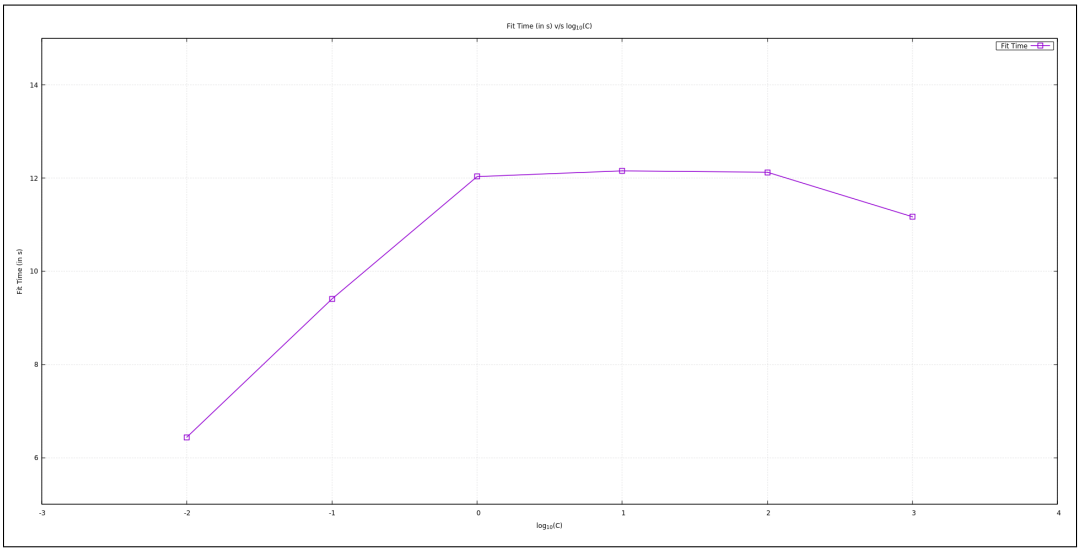

Figure 8: *Fit time (in s) vs C in Logistic Regression*

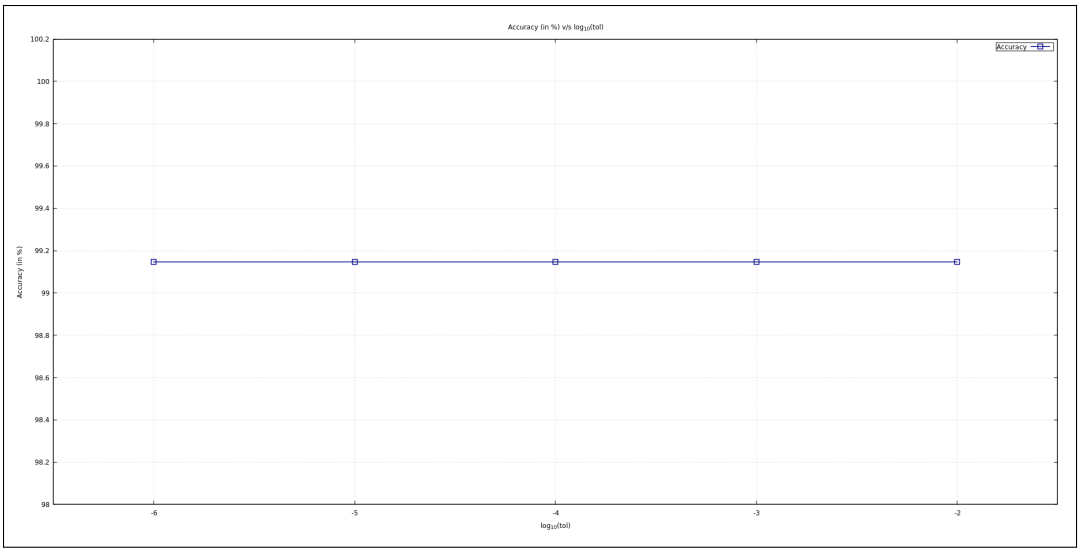

Figure 9: *Accuracy vs tol in Logistic Regression*

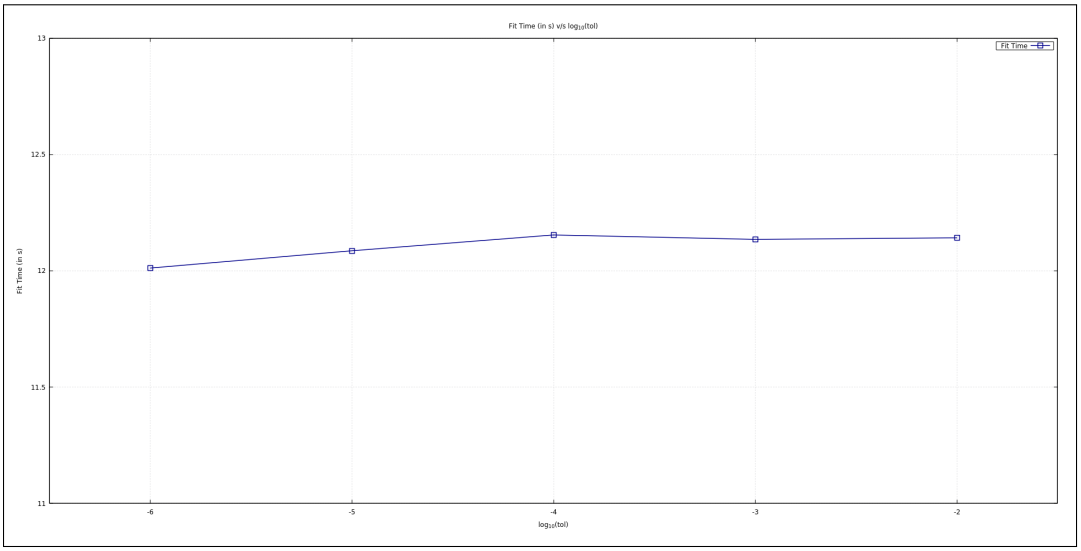

Figure 10: *Fit Time (in s) vs tol in Logistic Regression*

3.3 Data Tables

All parameters except the varying hyperparameters were kept constant (e.g. Number of Iterations)

Fit_Time (in s)	Predict_Time (in s)	Loss	Model_Size	C	Tol	NoIter	Accuracy
2.88199	1.66759	SqHinge	8936	0.01	1e-3	1000	97.7
3.7577	1.3283	SqHinge	8936	0.1	1e-3	1000	98.72
5.11249	1.2057	SqHinge	8929	1	1e-3	1000	99.0669
4.42822	1.43535	SqHinge	8929	10	1e-3	1000	99.1755
4.322468	1.43839	SqHinge	8929	100	1e-3	1000	99.057
4.4368	1.38174	SqHinge	8930	1000	1e-3	1000	98.972

Figure 11: *C variation in LinearSVC*

Fit_Time (in s)	Predict_Time (in s)	Loss	Model_Size	C	Tol	NoIter	Accuracy
4.6943	1.15008	SqHinge	8929	10	1e-2	1000	99.1695
4.42822	1.43535	SqHinge	8929	10	1e-3	1000	99.1755
4.4376	1.477711	SqHinge	8929	10	1e-4	1000	99.156
4.53716	1.22951	SqHinge	8929	10	1e-5	1000	99.1935
4.65872	1.16621	SqHinge	8929	10	1e-6	1000	99.1845

Figure 12: *Tol Variation in LinearSVC*

Fit_Time (in s)	Predict_Time (in s)	Model_Size	C	Tol	NoIter	Accuracy
6.4423	1.530	8966	0.01	1e-4	100	93.515
9.4072	1.1504	8966	0.1	1e-4	100	97.683
12.031	1.0091	8959	1.0	1e-4	100	98.75
12.154	1.3925	8959	10	1e-4	100	99.148
12.124	1.022	8959	100	1e-4	100	99.265
11.169	1.3748	8960	1000	1e-4	100	99.292

Figure 13: *C variation in Logistic Regression*

Fit_Time (in s)	Predict_Time (in s)	Model_Size	C	Tol	NoIter	Accuracy
12.14177	1.0250	8959	10	1e-2	100	99.1475
12.135	1.0310	8959	10	1e-3	100	99.1475
12.154	1.3925	8959	10	1e-4	100	99.1475
12.0860	1.4011	8959	10	1e-5	100	99.1475
12.01166	1.019	8959	10	1e-6	100	99.1475

Figure 14: *Tol variation in Logistic Regression*

3.4 Outcomes

1. It can be observed that accuracy and fit time reach optimal values for medium values of C in LinearSVC.
2. Tolerance doesn't affect accuracy as well as fit time much in LinearSVC.
3. It is observed that accuracy increases with increase in the value of C and saturates for higher values of C in Logistic regression. On the other hand, fit time first increases and decreases for high values of C in LR.
4. For logistic regression, accuracy and fit time remain almost constant with varying values of tolerance.
5. For LinearSVC, the Loss function doesn't have a major impact on Accuracy.
6. For LinearSVC, L1 penalty reaches almost same accuracy as L2 but time taken to train the model is significantly higher.
