Assignment 01 : CS771A
Introduction to Machine Learning

Group name : Melbo2.0
Samarth Kumar (210911), Pratyush Amrit (210762), Siddhant Srivastava (211029),
Anjali Jangir (210144), Himanshu Yadav (210445), Saurav Kr. Gupta (210951)

Problem 1.1 (Companion Arbiter PUF). Melbo is constantly innovating to come up with PUFs that cannot be broken by ML attacks. Recall that an arbiter PUF is a chain of k multiplexers, each of which either swaps the lines or keeps them intact, depending on what is the challenge bit fed into that multiplexer. The multiplexers each have delays which are hard to replicate but consistent. Let t^u, t^l respectively denote the time for the upper and lower signals to reach the finish line and let $\Delta \stackrel{\text{def}}{=} t^u - t^l$ denote the difference in the timings. Note that Δ can be negative or positive. Assume that $t^u = t^l$ never happens i.e., Δ is never exactly zero. If the upper signal reaches the finish line first i.e. $\Delta < 0$, the response is 0 else if the lower signal reaches first i.e. $\Delta > 0$, the response is 1.

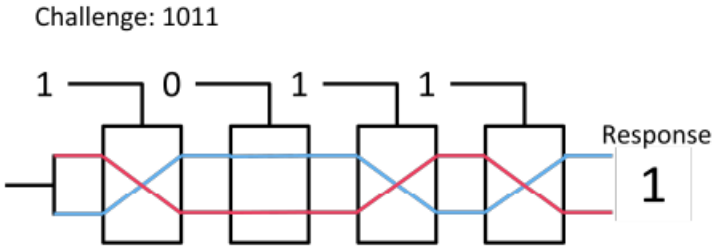

Figure 1: A simple arbiter PUF with 4 multiplexers

Melbo came up with an idea of creating a PUF using multiple arbiter PUFs and decided to call it a **Companion ArbiteR PUF** (CAR-PUF for short). 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.

Melbo thinks that with multiple PUFs whose delays are being compared against a secret threshold value, it is very difficult for a linear machine learning model to can predict the responses if given a few thousand challenge-response pairs. Your job is to prove Melbo wrong! You will do this by showing that there does exist a linear model that can perfectly predict the responses of a CAR-PUF and this linear model can be estimated fairly accurately if given enough challenge-response pairs (CRPs).

Preprint. Under review.

1 Mathematical Proof:

For the i^{th} PUF, let:

- c_i be the binary input given to the Multiplexer
- p_i be the delay for a signal starting at the top and exiting at the top
- q_i be the delay for a signal starting at the bottom and exiting at the bottom
- r_i be the delay for a signal starting at the top and exiting at the bottom
- s_i be the delay for a signal starting at the bottom and exiting at the top
- t_i^u be the total time it takes for the signal to come out of the top
- t_i^l be the total time it takes for the signal to come out of the bottom.

We define $\Delta_i = t_i^u - t_i^l$, as the time delay between the top and bottom signals exiting the PUF. We also define $\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}$, $\beta_i = \frac{p_i - q_i - r_i + s_i}{2}$, $W_i = \alpha_i + \beta_i$, and $d_i = 1 - 2c_i$ to be used later.

For a PUR with N multiplexers, we can express the total delay as the Δ using the following equation:

$$\Delta = \alpha_0 x_o + W_1 x_1 + ... + W_{N-1} x_{N-1} + \beta_{N-1} = W^T x + b, \tag{1}$$

where $x_i = d_i d_{i+1} ... d_{N-1}$, and W and x are N-dimensional vectors and b is a scalar.

In the case of a CAR-PUF, we have two arbiter PUFs $landv$, with delays Δ_w and Δ_r respectively. This combination returns the following outputs:

- Output = 1, if $|\Delta_w - \Delta_r| > \tau$
- Output = 0, if $|\Delta_w - \Delta_r| \leq \tau$

Keeping in mind that τ is positive, we can re-write the above as the following:

- Output = 1, if $(\Delta_w - \Delta_r)^2 > \tau^2$
- Output = 0, if $(\Delta_w - \Delta_r)^2 \leq \tau^2$

This way, we do not have to worry about the modulus. Define:

$$\begin{aligned} h(x) &= |\Delta_w - \Delta_r|^2 - \tau^2 \\ &= (W_w^T x + b_w - W_r^T x - b_r)^2 - \tau^2. \end{aligned}$$

Now, $W \stackrel{\text{def}}{=} W_w - W_r$ and $b \stackrel{\text{def}}{=} b_w - b_r$

$$\begin{aligned} h(x) &= (W^T x + b)^2 - \tau^2. \\ &= (W^T x)^2 + 2bW^T x + b^2 - \tau^2. \\ &= (\sum_{i=0}^{N-1} W_i x_i)^2 + 2b(\sum_{i=0}^{N-1} W_i x_i) + b^2 - \tau^2. \end{aligned}$$

By opening the square, we are left with N terms of the form x_i^2, ${}^N C_2$ terms of the form $x_i x_j$,$\forall\ i \neq j;\ i, j \in Z^+$ and N terms of the form x_i Now, we know that all input terms c_i are either 0 and 1, which means that all x_i are either 1 or -1 (from the definition above). This means that squaring x_i would always give 1, meaning that all the corresponding coefficients can be grouped together with the constant bias term. We are left with a total of $N +^N C_2$ terms, plus the bias term that is a scalar constant.

We can now write $h(x) = W^T \phi(c) + b$, where $\phi(c)$ is $N +^N C_2$ dimensional vector $\equiv [x_0, x_1, .., x_{N-2}, x_{N-1}, x_0 x_1, x_0 x_2, ..., x_{N-1} x_{N-3}, x_{N-1} x_{N-2}]$, W is the corresponding vector of coefficients and b is a scalar. By such a definition, we have turned the problem into a $N +^N C_2$ dimensional linear equation. The output of the CAR-PUF will be given by $r = \frac{1 + sign(W^T \phi(c) + b)}{2}$.

We have a 32 dimensional input vector in our data, which means that our feature vector W would have 528 terms. This means our model would need to learn a total of 529 parameters.

2 Experimentation Outcomes:

We have experimented with various linear regression methods present in the SciKit-Learn library, the results of which are present below.

2.1 LinearSVC

The LinearSVC method has three main hyper-parameters, 'C', which is used to define the regularisation criteria, the 'tolerance', and the 'maximum-iterations'. We experimented with various different combinations for the same.

C					
c	**Tolerance**	**t_train**	**t_map**	**Accuracy**	**Max_iter**
0.01	0.0001	8.8383	0.9069	0.9865	5000
0.1	0.0001	19.3595	0.9092	0.9899	5000
1	0.0001	36.8823	0.9275	0.9919	5000
10	0.0001	30.3616	0.9095	0.993	5000
100	0.0001	30.8018	0.9242	0.9921	5000
1000	0.0001	30.2334	0.9276	0.9919	5000
10000	0.0001	30.2782	1.2263	0.992	5000

Table 1: For constant Tolerance = 0.0001, Maximum Iteration = 5000

Tolerance					
c	**Tolerance**	**t_train**	**t_map**	**Accuracy**	**Max_iter**
1	0.000001	39.2316	0.8960	0.9919	5000
1	0.00001	38.7014	0.88014	0.992	5000
1	0.0001	38.5295	0.8626	0.9918	5000
1	0.001	37.6143	0.8712	0.9919	5000
1	0.01	35.6326	0.9150	0.9919	5000
1	0.1	28.7293	0.8989	0.992	5000
1	1	17.8105	1.1681	0.9914	5000
1	10	5.7930	0.9307	0.9317	5000

Table 2: For constant C = 1, Maximum Iteration = 5000

Maximum Iteration					
c	**Tolerance**	**t_train**	**t_map**	**Accuracy**	**Max_iter**
1	0.0001	4.6848	0.8434	0.9621	10
1	0.0001	7.8134	0.8613	0.9813	100
1	0.0001	16.2590	1.3274	0.9911	1000
1	0.0001	37.8044	0.8831	0.9919	5000
1	0.0001	60.3608	1.3095	0.9919	10000

Table 3: For constant C=1, Tolerance = 0.0001

2.2 Logistic Regression

The logistic regression method also has the exam same hyperparameters as defined above. The results of the same are given below:

C					
c	**Tolerance**	**t_train**	**t_map**	**Accuracy**	**Max_iter**
0.01	0.0001	3.9045	0.9131	0.9635	5000
0.1	0.0001	4.0978	0.8899	0.9871	5000
1	0.0001	4.644	0.888	0.9907	5000
10	0.0001	4.7574	0.816	0.9922	5000
100	0.0001	5.4557	0.8217	0.9931	5000
1000	0.0001	6.6233	0.9909	0.9923	5000
10000	0.0001	9.1038	0.8983	0.9921	5000

Table 4: For constant Tolerance = 0.0001, Maximum Iteration = 5000

Tolerance					
c	**Tolerance**	**t_train**	**t_map**	**Accuracy**	**Max_iter**
100	0.000001	4.8461	1.0254	0.9931	5000
100	0.00001	5.2766	0.877	0.9931	5000
100	0.0001	5.6379	0.8144	0.9931	5000
100	0.001	4.828	0.9568	0.9931	5000
100	0.01	4.6018	0.9436	0.9931	5000
100	0.1	4.6721	0.9024	0.9931	5000
100	1	4.2143	0.9239	0.9931	5000
100	10	4.1037	0.9272	0.9923	5000

Table 5: For constant C = 100, Maximum Iteration = 5000

Maximum Iterations					
c	**Tolerance**	**t_train**	**t_map**	**Accuracy**	**Max_iter**
100	0.0001	5.025	0.8534	0.9931	100
100	0.0001	4.9711	9742	0.9931	500
100	0.0001	5.2439	0.8306	0.9931	1000
100	0.0001	5.1048	0.9654	0.9931	5000
100	0.0001	5.2524	0.8882	0.9931	10000

Table 6: For constant C=100, Tolerance = 0.0001

After thorough tuning and experimentation, the following hyperparameters were found to be optimal for the logistic regression model.

Regularization Constant (C) = 130
Tolerance (tol) = 0.0001

These values were selected based on their performance in minimizing the loss function and improving overall model accuracy. Please find the graphs corresponding to the above experimentation in the appendix section.

3 Appendix

3.1 Graphs

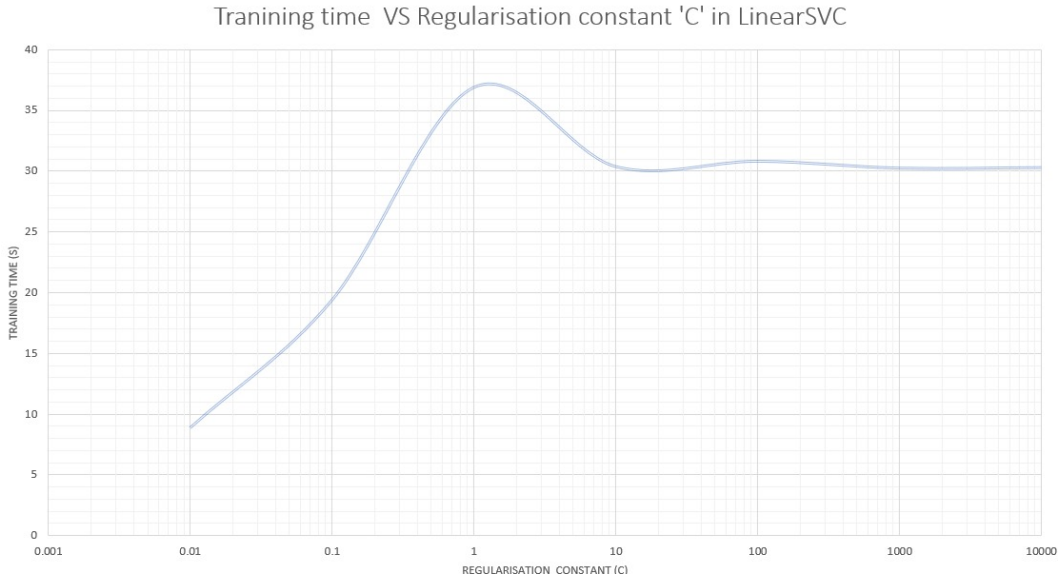

Figure 1: Training time increases with increase in C to a certain amount and then remains nearly contant for our model using LinearSVC

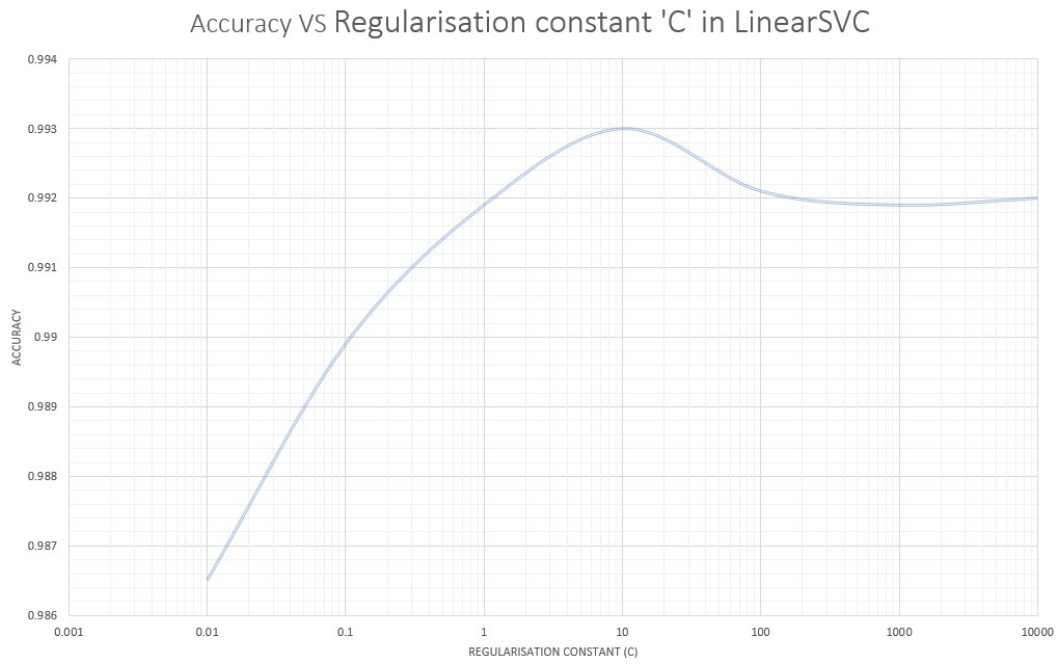

Figure 2: Accuracy attains it maximum value at around C = 10 for our model using LinearSVC

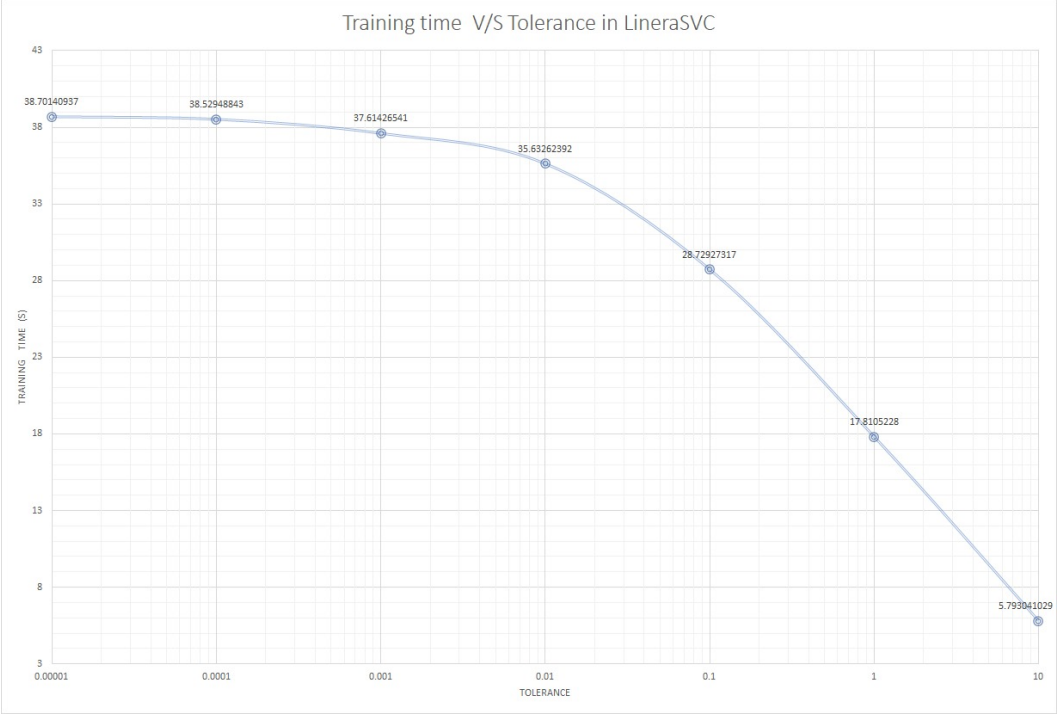

Figure 3: Training time decreases with increase in tolerance. This is because the model has to iterate for less number of times before reaching convergence.

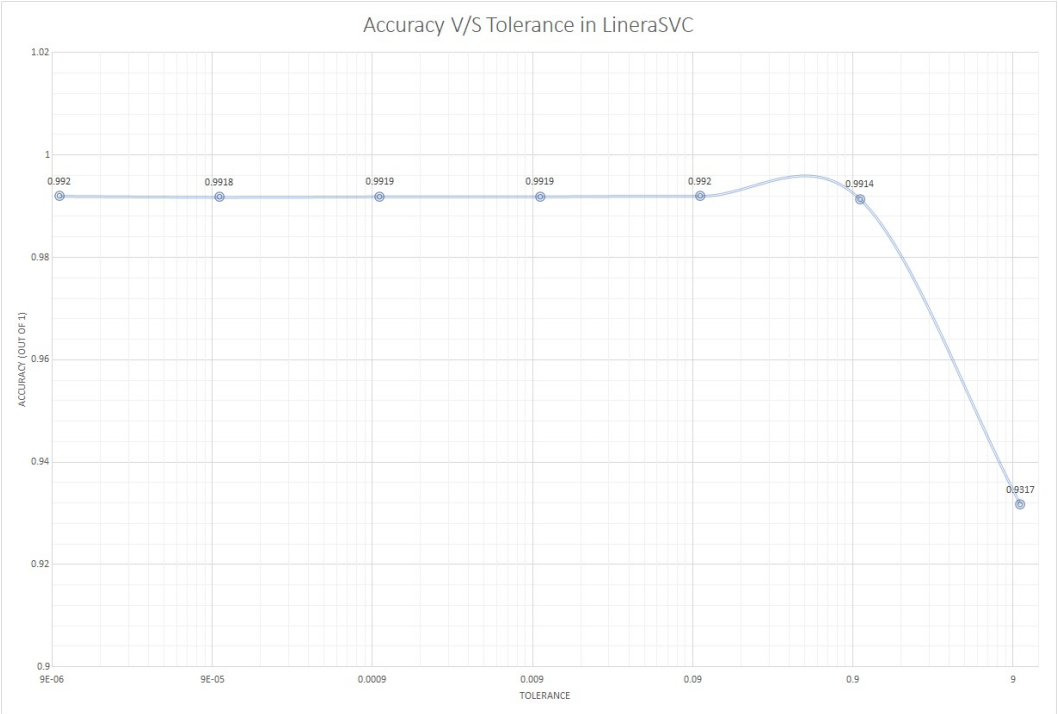

Figure 4: Accuracy increases with decrease in tolerance to a certain amount then remains constant

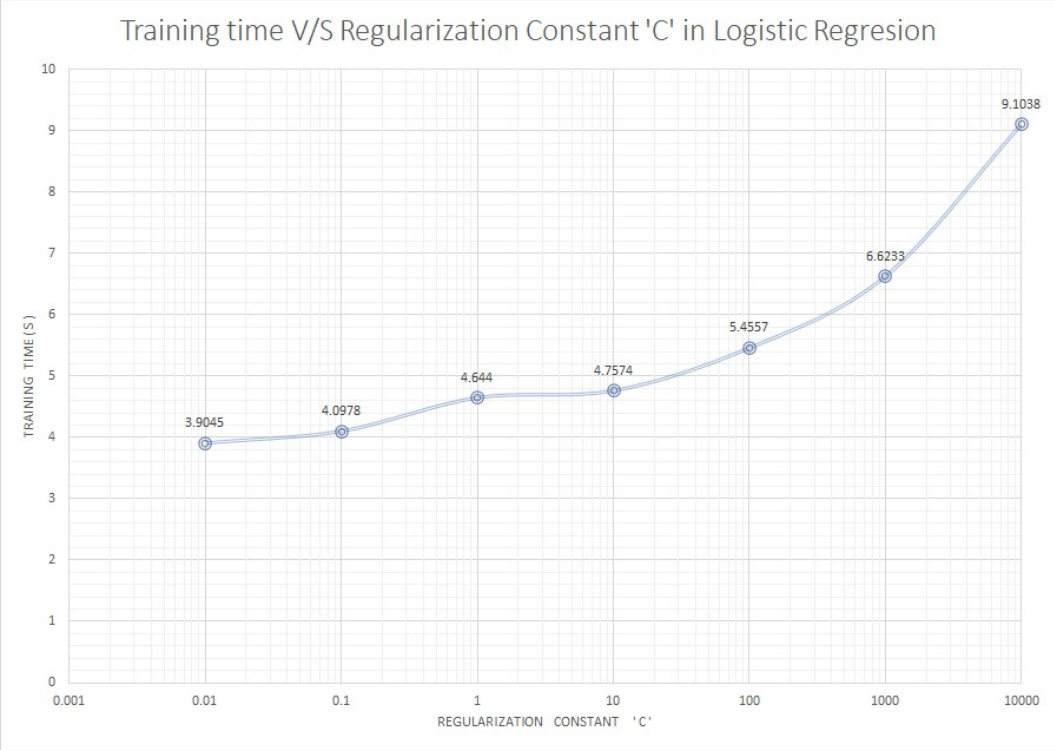

Figure 5: Training time keeps on increasing with increase in C for our model using Logistic Regression

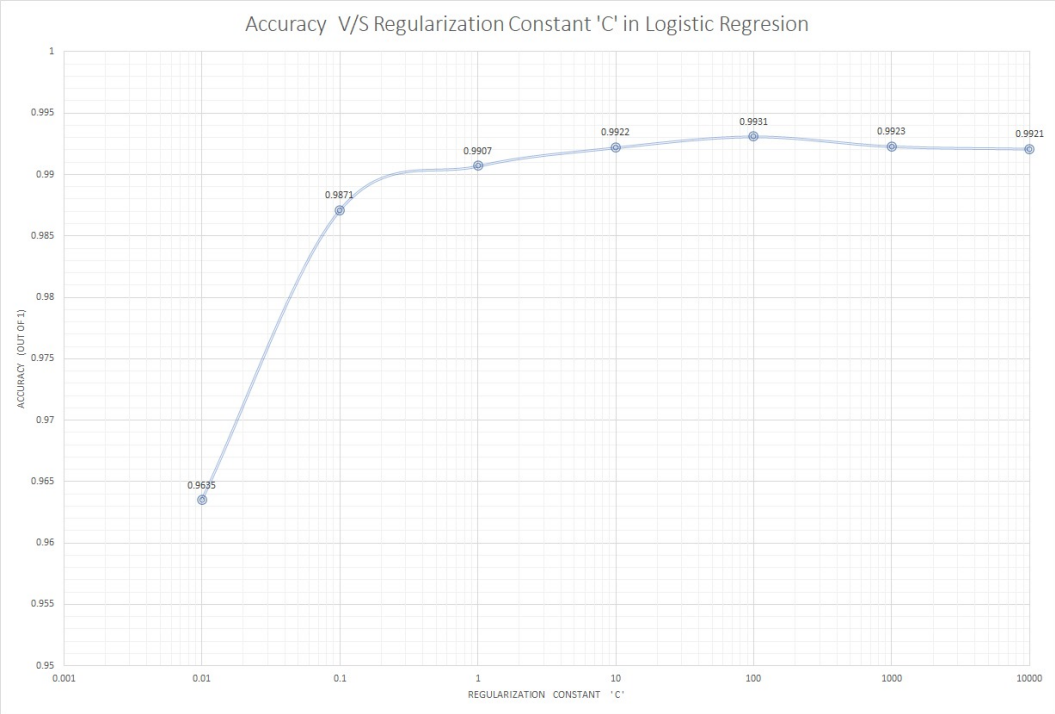

Figure 6: Accuracy attains it maximum value at around C = 100 for our model using Logistic Regression

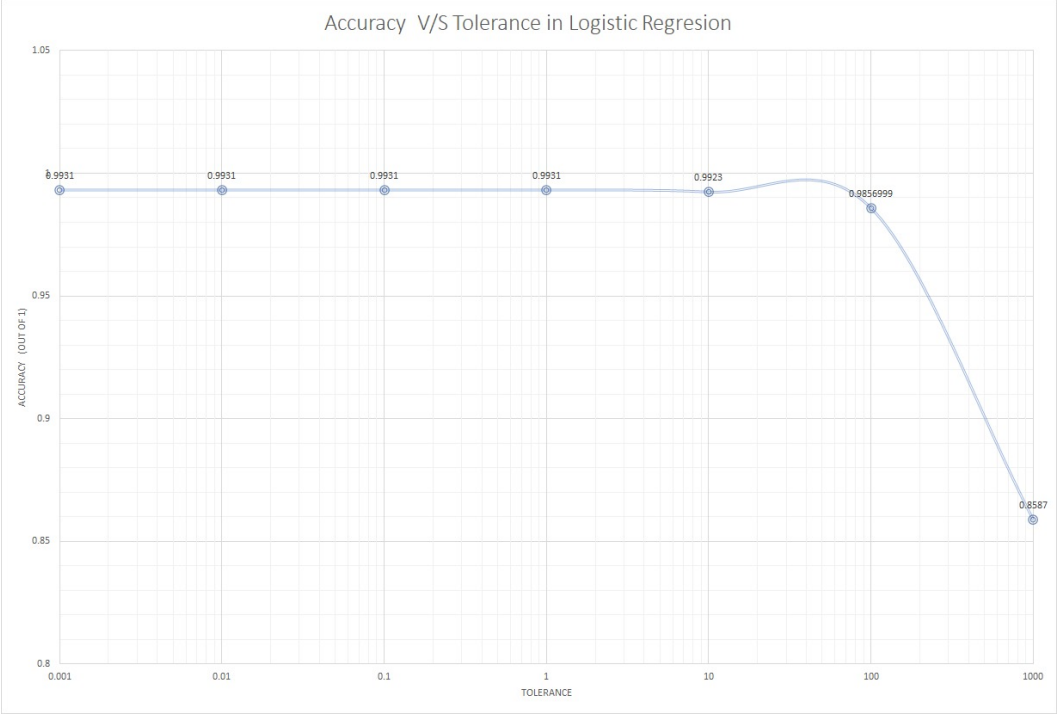

Figure 7: Accuracy increases with decrease in tolerance to a certain amount then remains constant

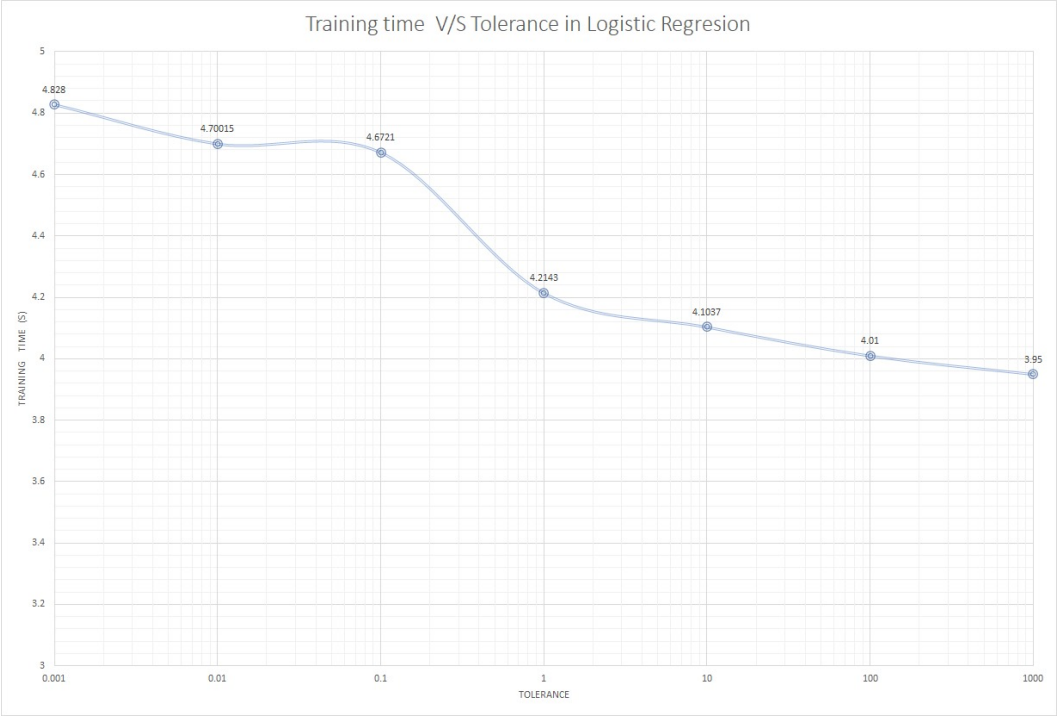

Figure 8: Training time decreases with increase in tolerance. This is because the model has to iterate for less number of times before reaching convergence