CS771 | Introduction to Machine Learning
Assignment 1 - Report

Lavesh Gupta
210562
laveshg21@iitk.ac.in

Ravi Kumar
210833
kravi21@iitk.ac.in

Ishan Agrawal
210455
ishanaag21@iitk.ac.in

Rahul Singh
210807
rahulsgh21@iitk.ac.in

Akash Kumar
210085
kakash21@iitk.ac.in

Sahil Tomar
210898
stomar21@iitk.ac.in

Abstract

In this assignment, we demonstrate the vulnerability of Melbo's Companion ArbiteR PUF (`CAR-PUF`) to linear models, providing a detailed mathematical derivation of a mapping function that breaks the security of `CAR-PUF`. Subsequently, we implement and analyze the linear model using `sklearn's LinearSVC` and `LogisticRegression`, exploring the impact of hyperparameters on training time and test accuracy to validate the susceptibility of `CAR-PUF` to linear modeling techniques.

1 Companion Arbiter PUF

1.1 Overview

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 $\delta := t^u - t^l$ 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

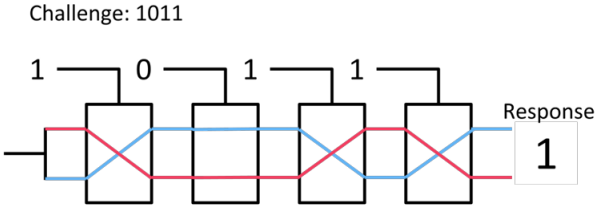


Figure 1: A simple arbiter PUF with 4 multiplexers

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).

1.2 Given Data

We have been provided with data from a CAR-PUF with 32-bit challenges. The training set consists of 40000 CRPs and the test set consists of 10000 CRPs. If we wish, we are allowed to create (held out/k-fold) validation sets out of this data in any way we like. If we properly encode the challenge as a 32-dim vector, then for any arbiter PUF, there exists a linear model that always predicts the correct response. Our job is to create a new feature vector from the 32-bit challenge so that a linear model applied to that feature vector exactly predicts the response for a CAR-PUF. However, a twist in the tale is that unlike the arbiter PUF case where the new feature vector was also 32-dimensional for 32-dimensional challenges, here the new feature vector may need to have a different dimensionality than the challenges.

2 Tasks

2.1 Task 1 : Mathematical Derivation

By giving a detailed mathematical derivation, show how a CAR-PUF can be broken by a single linear model. Give derivations for a map $\phi : \{0,1\}^{32} \rightarrow \mathbb{R}^D$ mapping a 32-bit 0/1-valued challenge to a D-dimensional feature vector (for some D > 0) so that for any CAR-PUF, there exists a D-dimensional linear model $\mathbf{W} \in \mathbb{R}^D$ and a bias term $b \in \mathbb{R}$ such that for all CRPs (**c**,r) with $\mathbf{c} \in \{0,1\}^{32}, r \in \{0,1\}$ we have

$$\frac{1+sign(\mathbf{W}^T\phi(\mathbf{c})+b)}{2} = r$$

Solution: Let's consider working of a single Arbiter PUF and let's denote the lag between top signal and bottom signal at i-th PUF by $\Delta_i = t_i^u - t_i^l$. So, all that matters is whether the top signal reaches first or not. Thus, all that matters is whether $\Delta_{31} < 0$ or not.

From the lecture slides we know that for a single Arbiter PUF requiring 32-bit challenges, we can write the final time lag as:

$$\Delta_{31} = w_0.x_0 + w_1.x_1 + w_2.x_2 + \ldots w_{31}.x_{31} + \beta_{31} \tag{1}$$

Given a challenge vector **c** we encode it to a vector **d** such that $d_i = 1 - 2\ c_i$ and $w_0 = \alpha_0$ and $w_i = \alpha_i + \beta_{i-1}$ are the PUF dependent (unique) coefficients. We then derive the vector **x** as

$$x_i = d_i.d_{i+1}.d_{i+2} \ldots d_{31}$$

Equation (1) can we re-written in vector form as as dot product of two vectors **w** and **b**

$$\Delta_{31} = [w_0, w_1, w_2, \ldots, w_{31}] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{31} \end{bmatrix} + [b] = \mathbf{w}^T\mathbf{x} + \mathbf{b}$$

$$\Delta_{31} = [w_0, w_1, w_2, \ldots, w_{31}, b] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{31} \\ 1 \end{bmatrix} = \widetilde{\mathbf{w}}^T \widetilde{\mathbf{x}}$$

In equation (2) **w** and **x** are 32 dimensional vectors, and to have an implicit bias term we add the bias term to model vector **w** and add a 1 to the encoded challenge vector **x** to account for the bias term.

$$\Delta_{31} = \mathbf{w}^T \mathbf{x} + b = [\mathbf{w}, b]^T [\mathbf{x}, \mathbf{1}] \tag{2}$$

The response of a single arbiter PUF (0,1 label) can be written as $\frac{1 + \text{sgn}(\Delta_{31})}{2}$

Companion Arbiter PUF (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.

Let (**u**,p), (**v**,q) be the two linear models that can exactly predict the outputs of the two arbiter PUFs sitting inside the CAR-PUF. Then $\mathbf{W}, b$ will depend on $\mathbf{u}, \mathbf{v}, p, q, \tau$ (PUF-specific constants) and then $\phi(\mathbf{c})$ depends only on c and does not use PUF-specific constants.

For working PUF:

$$\Delta_w = \mathbf{u}^T \mathbf{x} + p = \mathbf{m}^T \widetilde{\mathbf{x}} \qquad \mathbf{m}^T = [\mathbf{u}^T, p] \tag{3}$$

For reference PUF:

$$\Delta_r = \mathbf{v}^T \mathbf{x} + q = \mathbf{n}^T \widetilde{\mathbf{x}} \qquad \mathbf{n}^T = [\mathbf{v}^T, q] \tag{4}$$

For CAR-PUF:

$$\text{Response} = \frac{1 + \text{sgn}(|\Delta| - \tau)}{2}$$
$$\text{where } \Delta = \Delta_w - \Delta_r$$

We have to show that the response of the CAR-PUF can be modeled as a linear model $\frac{1+sign(\mathbf{W}^T \phi(\mathbf{c})+b)}{2}$ such that:

$$\frac{1 + \text{sgn}(|\Delta| - \tau)}{2} = \frac{1 + \text{sgn}(\mathbf{W}^T \phi(c) + b)}{2} = r \tag{5}$$

Simplifying the above expression leads us to:

$$\text{sgn}(|\Delta| - \tau) = \text{sgn}(\mathbf{W}^T \phi(c) + b) \tag{6}$$

Simplifying the LHS of equation (6):

Since, $|\Delta| + \tau > 0$

$$\text{sgn}(|\Delta| - \tau) = \text{sgn}((|\Delta| - \tau)(|\Delta| + \tau)) = \text{sgn}(\Delta^2 - \tau^2) \tag{7}$$

Comparing equation (6) and (7) we get:

$$\text{sgn}(\Delta^2 - \tau^2) = \text{sgn}(\mathbf{W}^T \phi(c) + b) \tag{8}$$

Using the linear models for working PUF and reference PUF from equation (3) and (4) respectively, we find the expression for $|\Delta|$ as

$$\Delta = \mathbf{m}^T \widetilde{\mathbf{x}} - \mathbf{n}^T \widetilde{\mathbf{x}} \tag{9}$$

Substituting in LHS of equation (8) :

$$\Delta^2 - \tau^2 = (\mathbf{m}^T\widetilde{\mathbf{x}} - \mathbf{n}^T\widetilde{\mathbf{x}})^2 - \tau^2$$

Opening the squared term:

$$\Delta^2 - \tau^2 = (\mathbf{m}^T\widetilde{\mathbf{x}})(\mathbf{m}^T\widetilde{\mathbf{x}}) + (\mathbf{n}^T\widetilde{\mathbf{x}})(\mathbf{n}^T\widetilde{\mathbf{x}}) - 2(\mathbf{m}^T\widetilde{\mathbf{x}})(\mathbf{n}^T\widetilde{\mathbf{x}}) - \tau^2 \tag{10}$$

Rewriting each term on the RHS of equation (10) as follows:

$$(\mathbf{m}^T\widetilde{\mathbf{x}})(\mathbf{m}^T\widetilde{\mathbf{x}}) = \sum_{i=0}^{32}\sum_{j=0}^{32} m_i m_j x_i x_j \tag{11}$$

$$(\mathbf{n}^T\widetilde{\mathbf{x}})(\mathbf{n}^T\widetilde{\mathbf{x}}) = \sum_{i=0}^{32}\sum_{j=0}^{32} n_i n_j x_i x_j \tag{12}$$

$$(\mathbf{m}^T\widetilde{\mathbf{x}})(\mathbf{n}^T\widetilde{\mathbf{x}}) = \sum_{i=0}^{32}\sum_{j=0}^{32} m_i n_j x_i x_j \tag{13}$$

Let us denote the constants in above equations as:

$$m_i m_j = M_{ij} \qquad n_i n_j = N_{ij} \qquad m_i n_j = P_{ij}$$

Substituting the renamed constants, and equation (11)-(13) into equation (10):

$$\Delta^2 - \tau^2 = \sum_{i=0}^{32}\sum_{j=0}^{32}(M_{ij} + N_{ij} - 2P_{ij})x_i x_j - \tau^2 \tag{14}$$

Now, we can write this as a dot product of linear model vector W and feature vector $\widetilde{\phi(x)}$ of dimension $33^2 = 1089$:

$$\sum_{i=0}^{32}\sum_{j=0}^{32}(M_{ij} + N_{ij} - 2P_{ij})x_i x_j - \tau^2 = \sum_{i=0}^{33^2-1} W_i\phi(\widetilde{x})_i - \tau^2 = \mathbf{W}^T\widetilde{\phi(x)} \tag{15}$$

So, we get our feature map having D = 1089 as follows:

$$\widetilde{\phi(x)} = \sum_{i=0}^{32}\sum_{j=0}^{32} x_i x_j \tag{16}$$

Reducing the dimensionality of feature map:

- Since $x_i xj = x_j x_i$, we can only take those entries for whom $i \leq j$ and this would reduce the dimensionality to $D^{'} = 1089 - 528 = 561$ i.e. lower triangular matrix in equation (18)
- Also, since $x_i = \pm 1$, so the product $x_i.x_i = 1$ (i.e. when $i = j$) always for all $0 \leq i \leq 32$, so these entries would contribute to the bias term of the model i.e. diagonal of matrix in equation (18). Hence we can further reduce our dimensionality to $D^* = 561 - 33 = 528$

Hence our final feature vector having dimension **D = 528** is as follows:

$$\widetilde{\phi(x)} = \sum_{i=0}^{32}\sum_{\substack{j=0 \\ j>i}}^{32} x_i x_j \tag{17}$$

The **dimension reduced feature map** (D = 528) can be viewed as the **upper triangular matrix** in the below equation (18)

$$\widetilde{\phi(x)} = \begin{pmatrix} x_0x_0 & x_0x_1 & x_0x_2 & \dots & x_0x_{32} \\ x_1x_0 & x_1x_1 & x_1x_2 & \dots & x_1x_{32} \\ x_2x_0 & x_2x_1 & x_2x_2 & \dots & x_2x_{32} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{32}x_0 & x_{32}x_1 & x_{32}x_2 & \dots & x_{32}x_{32} \end{pmatrix} \tag{18}$$

Hence the dimensionality of feature map is **D = 528**

The bias term b in equation (6) becomes: $b = \sum_{i=0}^{32}(M_{ii} + N_{ii} - 2P_{ii}) - \tau^2$

2.2 Task 2 : Results Of Code

The final results of the code are as follows:

Table 1: Results

D	t_train	t_map	Misclassification Rate
528.0	1.7200	0.07186	0.0069

The python code for `my_fit()` function we used is as follows:

```
def my_fit( X_train, y_train ):
        features = my_map(X_train)
        labels = y_train
        hyperparameters = {
            'C': 100,
            'max_iter': 500,
            'penalty': 'l2',
            'solver': 'lbfgs',
            'tol': 0.01
        }
        model = LogisticRegression(**hyperparameters)
        model.fit(features, labels)
        w = model.coef_.flatten()
        b = model.intercept_[0]
        return w, b
```

The python code for `my_map()` function we used is as follows:

```
def my_map( X ):
    X = 2*X -1
    X = np.flip(np.cumprod(np.flip(X, axis = 1), axis = 1), axis = 1)
    X = np.hstack((X, np.ones((X.shape[0], 1))))
    num_columns = X.shape[1]
    feat = np.empty((X.shape[0], num_columns * (num_columns - 1) // 2))
    idx = 0
    for i in range(num_columns):
        feat[:, idx:idx+num_columns-i-1] = X[:, i+1:] * X[:, i][:, np.newaxis]
        idx += num_columns - i - 1
    return feat
```

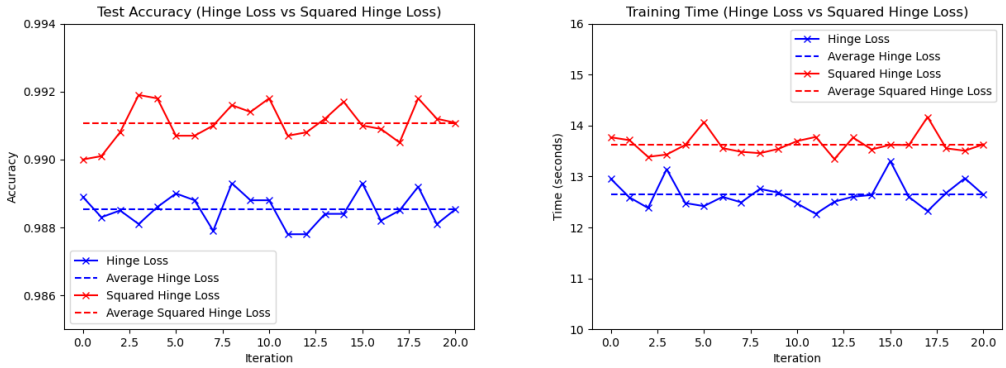
2.3 Task 3 : Varying Hyperparameters

Report outcomes of the following experiments with both the `sklearn.svm.LinearSVC` and `sklearn.linear_model.LogisticRegression` methods when used to learn the linear model. In particular, report how various `hyperparameters` affected training time and test accuracy using tables and/or charts with both `LinearSVC` and `LogisticRegression` methods.

2.3.1 Changing the loss hyperparameter in LinearSVC (hinge vs squared hinge)

Table 2: Changing the `loss` hyperparameter in LinearSVC

Loss Hyper-parameter	Training time (in sec)	Test accuracy (in %)
Hinge	12.6421s	0.9885
Squared Hinge	13.6299s	0.9910

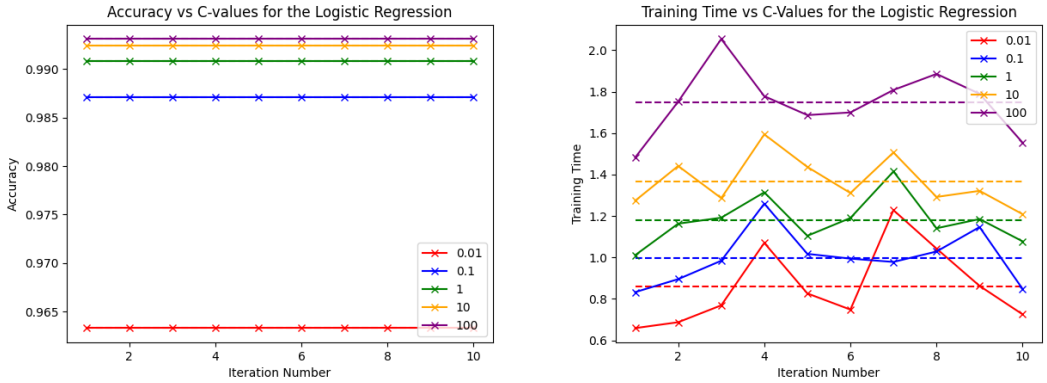


(a) Accuracy Comparison (b) Time Comparison

Figure 1: Changing the `loss` hyperparameter in LinearSVC

2.3.2 Setting C in LinearSVC and LogisticRegression to high/low/medium values.

For LogisticRegression:



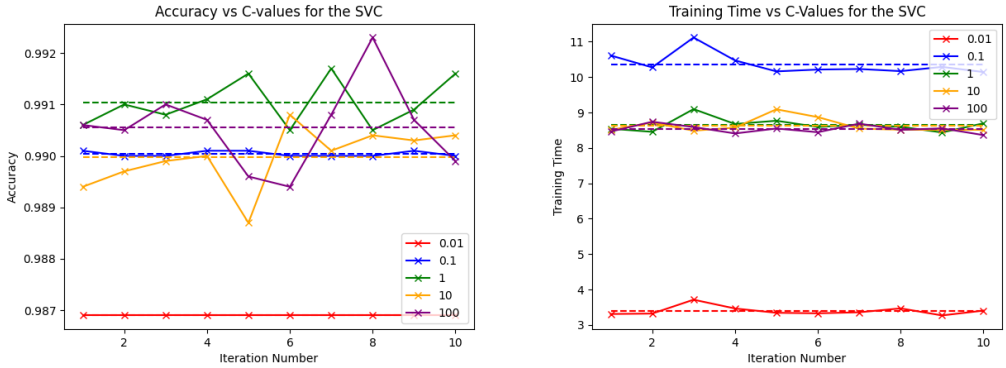
(a) Test Accuracy Comparison (b) Training Time Comparison

Figure 2: Changing the `C` hyperparameter in LogisticRegression

Table 3: Changing the `C` hyperparameter in LogisticRegression

C Hyper-parameter	Training time (in sec)	Test accuracy (in %)
0.01	1.557846864	0.9633
0.1	1.704299834	0.9871
1	1.855569939	0.9908
10	2.413217226	0.9924
100	2.818143446	0.9931

For LinearSVC:



(a) Test Accuracy Comparison (b) Training Time Comparison

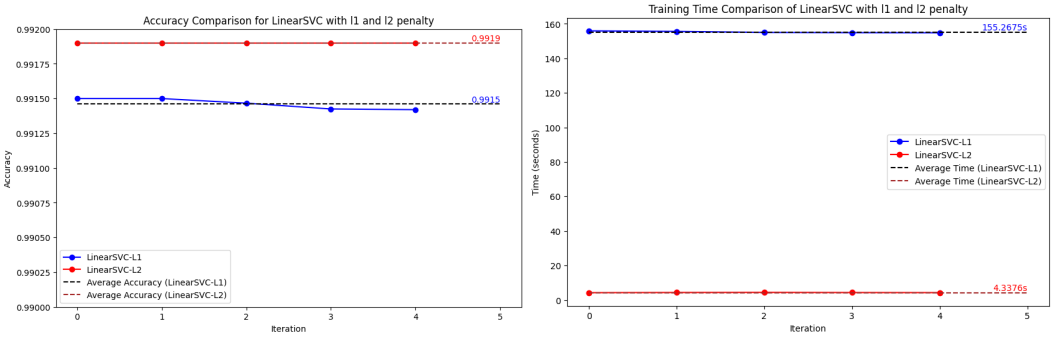
Figure 3: Changing the `C` hyperparameter in LinearSVC

Table 4: Changing the `C` hyperparameter in LinearSVC

C Hyper-parameter	Training time (in sec)	Test accuracy (in %)
0.01	5.5465	0.9869
0.1	16.2141	0.9900
1	13.5586	0.9909
10	13.4085	0.9902
100	13.1968	0.9906

2.3.3 Changing the penalty (regularization) hyperparameter in LinearSVC and LogisticRegression (l2 vs l1)

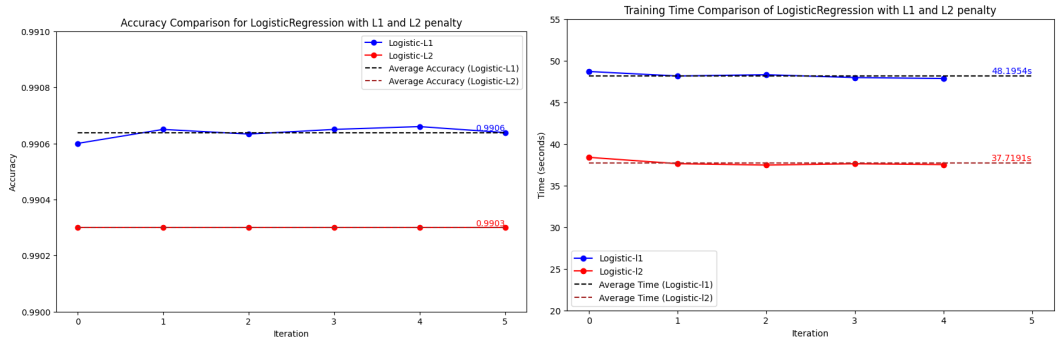
For LinearSVC:



(a) Test Accuracy Comparison (b) Training Time Comparison

Figure 4: Changing the `penalty` hyperparameter in Linear SVC

For LogisticRegression:

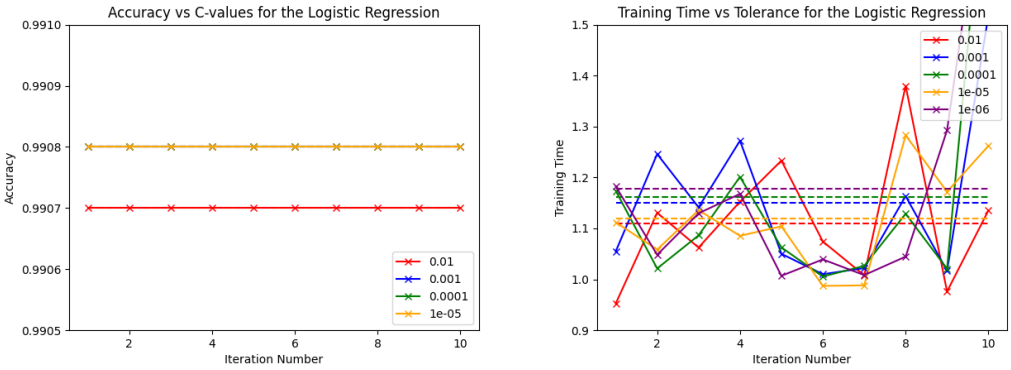


(a) Test Accuracy Comparison (b) Training Time Comparison

Figure 5: Changing the `penalty` hyperparameter in LogisticRegression

2.3.4 Changing tol in LinearSVC and LogisticRegression to high/low/medium values.

For LogisticRegression:



(a) Test Accuracy Comparison (b) Training Time Comparison

Figure 6: Changing the `tol` hyperparameter in LogisticRegression

For LinearSVC:

Table 5: Changing the `penalty` hyperparameter in LinearSVC

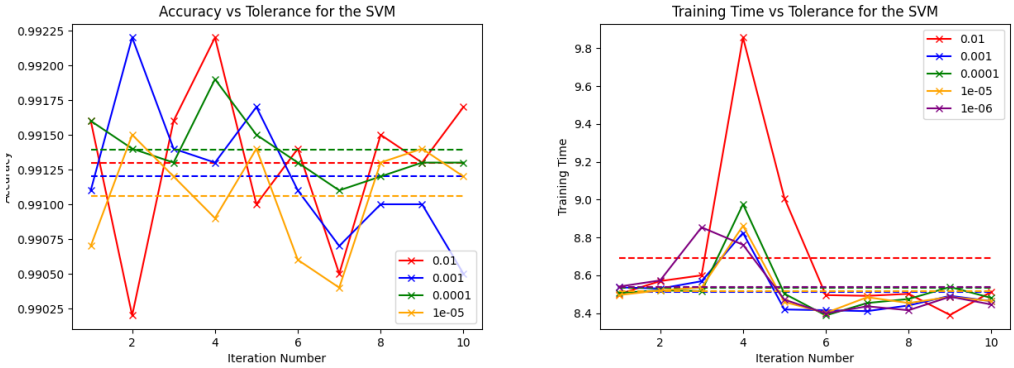
Penalty Hyperparameter	Training time (in sec)	Test accuracy (in %)
l1	155.2675s	0.9914
l2	4.3375s	0.9919

Table 6: Changing the `penalty` hyperparameter in LogisticRegression

Penalty Hyperparameter	Training time (in sec)	Test accuracy (in %)
l1	48.1954	0.9906
l2	37.7191	0.9903

Table 7: Changing the `tol` hyperparameter in LogisticRegression

tol Hyper-parameter	Training time (in sec)	Test accuracy (in %)
1e-7	1.842732588	0.9908
1e-6	2.180858064	0.9908
1e-5	1.942355638	0.9908
1e-4	1.838756612	0.9908
1e-3	1.931054364	0.9908



(a) Test Accuracy Comparison

(b) Training Time Comparison

Figure 7: Changing the `tol` hyperparameter in LinearSVC

Table 8: Changing the `tol` hyperparameter in LinearSVC

tol Hyper-parameter	Training time (in sec)	Test accuracy (in %)
1e-7	15.29431893	0.99102
1e-6	15.31693636	0.99134
1e-5	16.0118873	0.99104
1e-4	16.36328237	0.9912
1e-3	16.0331767	0.99124