CS771
INTRODUCTION TO MACHINE LEARNING

Assignment 1

Submitted By:
Group Name:- Thala Thala One

Roll No.	Member
210653	Navankur Shrotriya
210877	Roy Shivam Ram Shreshtth
210161	Anubhav Bairoliya
210045	Abhishek Raj
210127	Amrit Kumar
211043	Snehal Kumar Agrawal

1 Problem 1.1

By giving a detailed mathematical derivation (as given in the lecture slides), show how a CAR-PUF can be broken by a single linear model. Give derivations for a map $\phi : \{0,1\}^{32} \to \mathbb{R}^D$ mapping 32-bit 0/1-valued challenge vectors to D-dimensional feature vectors (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 $c \in \{0,1\}^{32}$, $r \in \{0,1\}$, we have:

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

Answer:

Here are the calculations for individual puffs of CAR-PUF. Here, we can see that the time gap can be expressed as a linear combination of cumulative products of $d_1, \ldots, d_{32}$ where each d_i is $1 - 2c_i$. Hence, we can express the linear model as $\sum_{i=1}^{32} W_i x_i + b$ where x_i is $d_{32} \cdot d_{31} \cdots d_i$, hence it can be expressed as a linear combination of some function of C. We will use this result to obtain a general linear equation for CAR-PUF.

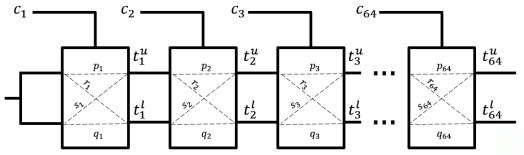

Figure 1: PUF

$$\begin{aligned}
\Delta_2 &= (1 - c_2) \cdot (t_1^u + p_2 - t_1^l - q_2) + c_2 \cdot (t_1^l + s_2 - t_1^u - r_2) \\
&= (1 - 2c_2) \cdot \Delta_1 + (q_2 - p_2 + s_2 - r_2) \cdot c_2 + (p_2 - q_2) \\
\Delta_2 &= \Delta_1 \cdot d_2 + \alpha_2 \cdot d_2 + \beta_2 \qquad (1) \\
& \qquad (2)
\end{aligned}$$

$$\begin{aligned}
\Delta_1 &= \alpha_1 \cdot d_1 + \beta_1 \quad \text{(since } \Delta_0 = 0\text{)} \qquad (3) \\
\Delta_2 &= \alpha_1 \cdot d_2 \cdot d_1 + (\alpha_2 + \beta_1) \cdot d_2 + \beta_2 \\
\Delta_3 &= \alpha_1 \cdot d_3 \cdot d_2 \cdot d_1 + (\alpha_2 + \beta_1) \cdot d_3 \cdot d_2 + (\alpha_3 + \beta_2) \cdot d_3 + \beta_3 \\
\Delta_4 &= \alpha_1 \cdot d_4 \cdot d_3 \cdot d_2 \cdot d_1 + (\alpha_2 + \beta_1) \cdot d_4 \cdot d_3 \cdot d_2 \\
&\quad + (\alpha_3 + \beta_2) \cdot d_4 \cdot d_3 + (\alpha_4 + \beta_3) \cdot d_4 + \beta_4 \qquad (4)
\end{aligned}$$

Here are the calculations of CAR-PUF. Here we have two arbiter PUFs and their inputs are 32 dimensions. Here our variates are $x_1, x_2, \ldots, x_{32}$ are the same as derived in arbiter-puff. Now we will get a proper transformation.

$$\begin{aligned}
\Delta_w &= \mathbf{u}^T\mathbf{x} + p \\
\Delta_r &= \mathbf{v}^T\mathbf{x} + q \\
|\Delta_w - \Delta_r| &= |\mathbf{u}^T\mathbf{x} + p - \mathbf{v}^T\mathbf{x} - q| \\
|\Delta_w - \Delta_r| &= |(\mathbf{u} - \mathbf{v})^T\mathbf{x} + (p - q)| \qquad (5)
\end{aligned}$$

Given, If $|\Delta_w - \Delta_r| \leq \tau$, then $r = 0$, and if $|\Delta_w - \Delta_r| > \tau$, then $r = 1$

Let our model be $\mathbf{M}$,

$$\mathbf{M} = (\Delta_w - \Delta_r)^2 - \tau^2 \qquad (6)$$

$$\mathbf{M} = (\mathbf{u}^T\mathbf{x} - \mathbf{v}^T\mathbf{x} + p - q)^2 - \tau^2 \qquad (7)$$

Substituting $\mathbf{a}^T = \mathbf{u}^T - \mathbf{v}^T$ and $t = p - q$ in (7), we get:

$$\mathbf{M} = (\mathbf{a}^T\mathbf{x} + t)^2 - \tau^2 \qquad (8)$$

We can expand equation (8) as follows:

$$\mathbf{M} = \left(\sum_{i=1}^{32} a_i x_i + t\right)^2 - \tau^2$$

since x_i is the cumulative product of d_i's which are either 1 or -1, hence $x_i^2 = 1$

$$\mathbf{M} = \sum_{i=1}^{32} a_i^2 + 2 \sum_{1 \leq i < j \leq 32} a_i a_j x_i x_j \quad + 2\sum_{i=1}^{32} t a_i x_i + t^2 - \tau^2$$

$$\begin{aligned}
\text{Let } W &= [2ta_1 \quad \cdots \quad 2ta_{32} \quad 2a_1a_2 \quad 2a_1a_3 \quad \cdots \quad 2a_{31}a_{32}] \\
X &= [x_1 \quad \cdots \quad x_{32} \quad x_1x_2 \quad x_1x_3 \quad \cdots \quad x_2x_3 \quad \cdots \quad x_{31}x_{32}] \\
b &= t^2 - \tau^2 + \sum_{i=1}^{32} a_i^2
\end{aligned}$$

So, our model becomes:

$$\begin{aligned}
\mathbf{M} &= \mathbf{W}^T\mathbf{X} + b \\
\text{where, } \mathbf{W} &\in \mathbb{R}^{528}, \quad b \in \mathbb{R} \\
\frac{1 + \text{sign}(\mathbf{W}^T\mathbf{X} + b)}{2} &= r \\
\text{So, } r = 0 &\implies \mathbf{W}^T\mathbf{X} + b < 0 \\
r = 1 &\implies \mathbf{W}^T\mathbf{X} + b > 0
\end{aligned}$$

So, we use the mapping $\phi(\mathbf{c})$:-

$$\phi(\mathbf{c}) = [x_1 \quad \cdots \quad x_{32} \quad x_1x_2 \quad x_1x_3 \quad \cdots \quad x_2x_3 \quad \cdots \quad x_{31}x_{32}] \qquad (9)$$

$$\text{where } \mathbf{x}_i = d_{32} \times d_{31} \cdots \times d_i, \quad \text{here } d_j = 1 - 2c_j \text{ for all } j \qquad (10)$$

2 Problem 1.2

Write code to solve this problem by learning the linear model **W**, b using the training data. You are allowed to use any linear classifier formulation available in the sklearn library to learn the linear model (i.e. you need not write a solver yourself). For instance, you may use LinearSVC, LogisticRegression, RidgeClassifier etc. However, the use of non-linear models is not allowed. Submit code for your chosen method in `submit.py`. Note that your code will need to implement at least 2 methods namely

(a) `my_map()` that should take test challenges and map each one of them to a D-dimensional feature vector.
Answer:

```
def my_map( X ):
################################
#  Non Editable Region Ending  #
################################

  # Use this method to create features.
  # It is likely that my_fit will internally call my_map to create
    features for train points
    d = X.shape[0]
    u = np.ones((d,1))
    X = np.concatenate((X, u), axis=1)
    n = X.shape[1]
    n = np.shape(X)[1]
    X = np.cumprod(np.flip(2 * X - 1, axis=1), axis=1)
    feat = khatri_rao(X.T, X.T).T
    upp_mat = np.triu(np.ones((n, n)), k=0)-np.eye(n)
    u1d = upp_mat.flatten().astype(bool)
    feat = feat[:,u1d]

    return feat
```

Listing 1: Python example

(b) `my_fit()` that should take train CRPs and learn the linear model by invoking an sklearn routine. It is likely that `my_fit()` will internally call `my_map()`.
Answer:

```
def my_fit( X_train, y_train ):
################################
#  Non Editable Region Ending  #
################################

  # Use this method to train your model using training CRPs
  # X_train has 32 columns containing the challeenge bits
  # y_train contains the responses

  # THE RETURNED MODEL SHOULD BE A SINGLE VECTOR AND A BIAS TERM
  # If you do not wish to use a bias term, set it to 0
    X_train = my_map(X_train)
    clf = LogisticRegression(C = 1.0, tol=1e-8)
    clf.fit(X_train, y_train)
    w = clf.coef_[0]  # Access the coefficients via the named_steps
     attribute
    b = clf.intercept_[0]

    return w, b
```

Listing 2: Python example

3 Problem 1.3

Report outcomes of 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. Report these experiments with both LinearSVC and LogisticRegression methods even if your own submission uses just one of these methods or some totally different linear model learning method (e.g. RidgeClassifier) In particular, you must report how at least 2 of the following affect training time and test accuracy:

(a) changing the `loss` hyperparameter in LinearSVC (hinge vs squared hinge)
Answer:

Loss Function	Accuracy	Training Time (s)
Squared Hinge	0.9919	4.20121
Hinge	0.9898	9.527

Table 1: Training Time and Accuracy for different loss functions(LinearSVC)

(b) setting `C` in LinearSVC and LogisticRegression to high/low/medium values
Answer:

Table 2: Training Time, t_map, and Test Accuracy for Different Values of C in **LinearSVC**

C	Training Time (s)	t_map	Test Accuracy
0.0001	2.97381838	0.120711	0.8804
0.001	3.5230303	0.118129	0.9596
0.01	3.87091934	0.107786	0.9869
0.3	4.37542466	0.112981	0.9913
0.4	4.2166957	0.112737	0.9919
0.5	4.06920618	0.123145	0.9922
0.6	4.18189212	0.116033	0.9919
0.7	4.2432483	0.106067	0.9916
1	4.44756058	0.126101	0.9919
2	4.20772506	0.126135	0.9929
5	4.40065882	0.116169	0.9929
10	4.20768898	0.116203	0.993
15	4.541411	0.106237	0.9925
20	4.91790176	0.112271	0.9921
30	6.09859536	0.116305	0.992
100	32.11275218	0.116339	0.9919

Table 3: Training Time, t_map, and Test Accuracy for Different Values of C in **(LogisticRegression)**

C	Training Time (s)	t_map	Test Accuracy
0.0001	0.88093468	0.107738	0.8482
0.001	0.89022028	0.120711	0.9068
0.01	0.958469	0.118129	0.9633
0.3	1.09797868	0.107786	0.9898
0.4	1.12759544	0.112981	0.9902
0.5	1.12169092	0.112737	0.9902
0.6	1.04796044	0.112493	0.9903
0.7	1.04086234	0.112254	0.9903
0.8	1.07614676	0.112006	0.9905
1	1.09743736	0.111762	0.9907
2	1.05858564	0.111518	0.9918
5	1.043716	0.111274	0.9919
10	1.1456562	0.111031	0.9922
15	1.11484888	0.110787	0.9926
20	1.17433344	0.110543	0.9926
30	1.22685788	0.110299	0.9927
38	1.16692678	0.110055	0.9927
40	1.10627936	0.109812	0.9926
100	1.21570152	0.109568	0.9926

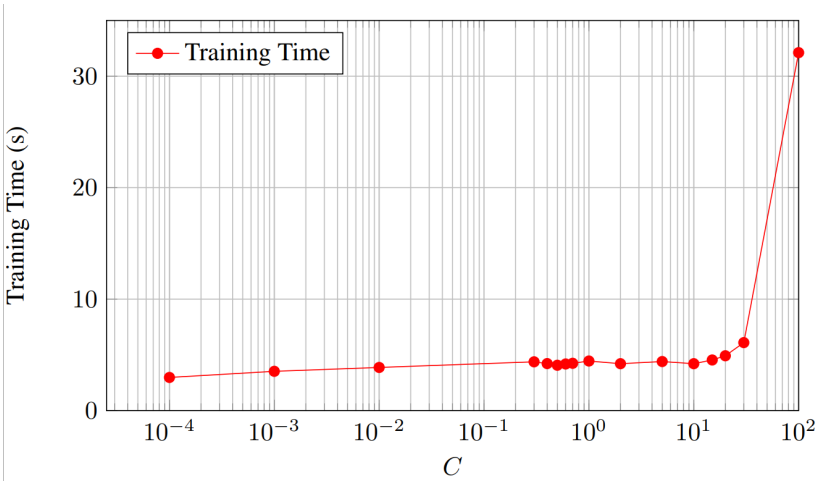

Figure 2: Training Time vs. C in LinearSVC

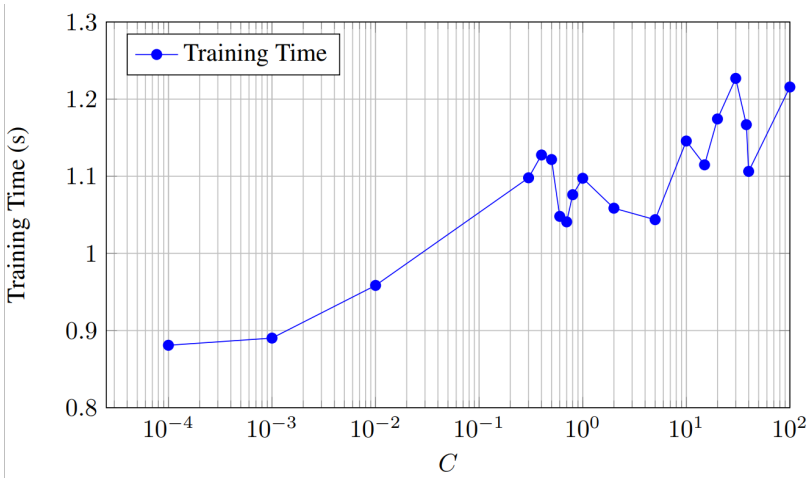

Figure 3: Training Time vs. C in LogisticRegression

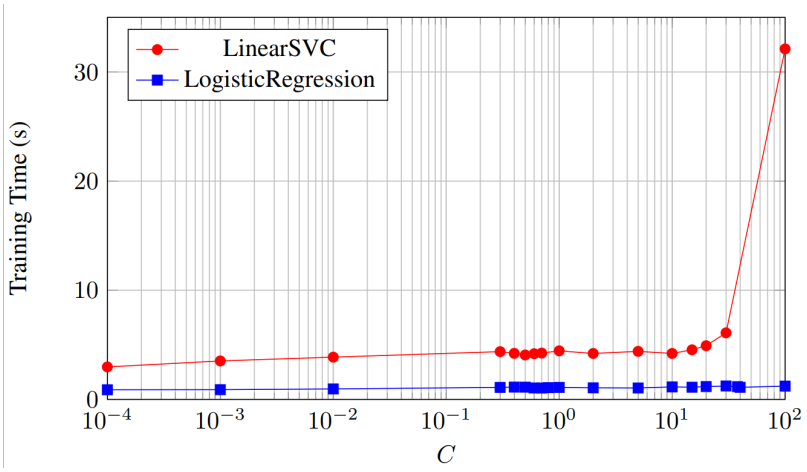

Figure 4: Training Time vs. C

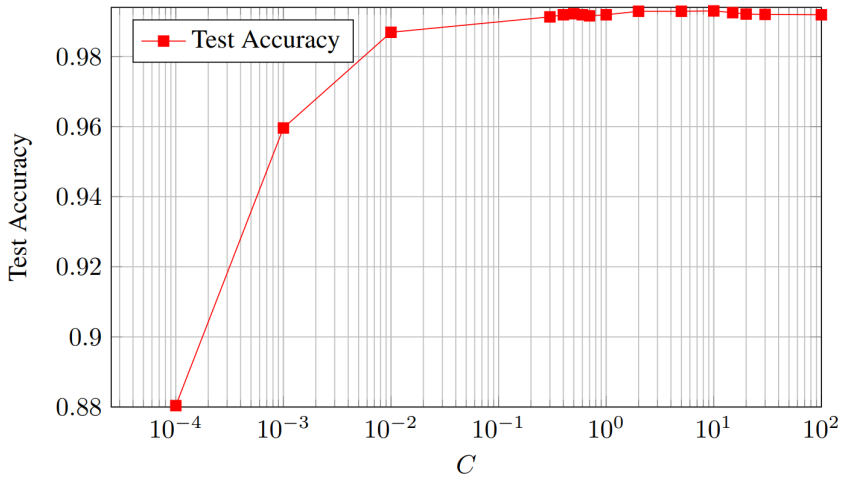

Figure 5: Test Accuracy vs. C in LinearSVC

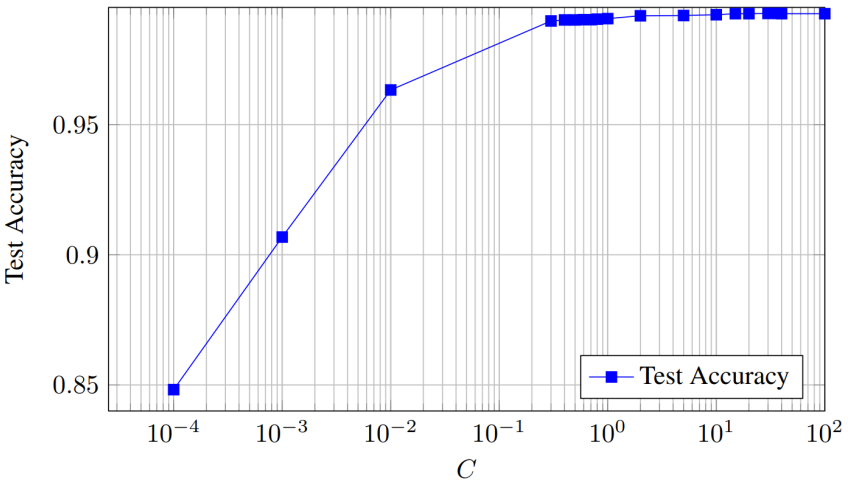

Figure 6: Test Accuracy vs. C in LogisticRegression

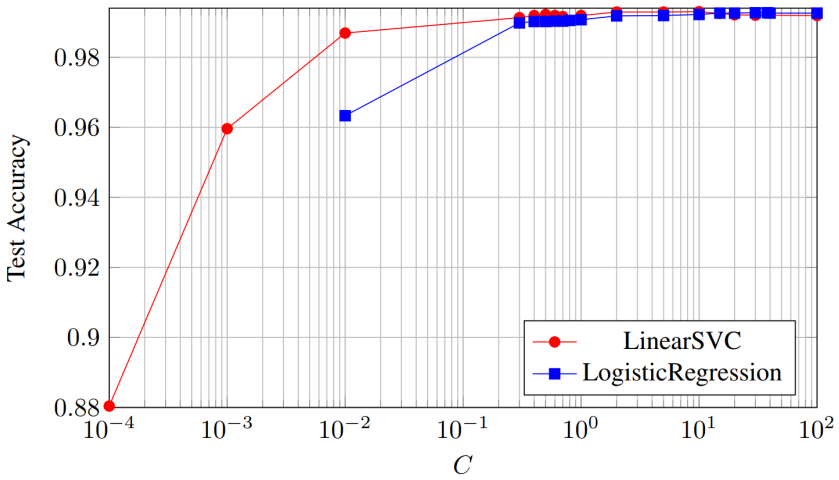

Figure 7: Test Accuracy vs. C

(c) Changing `tol` in LinearSVC and LogisticRegression to high/low/medium values
Answer:

Table 4: Training Time, t_map, and Test Accuracy for Different Values of Tolerance in **LinearSVC**

Tolerance	Training Time (s)	t_map	Test Accuracy
1.00×10^{-10}	4.44647998	0.113401	0.9919
1.00×10^{-6}	3.95488624	0.124509	0.9919
1.00×10^{-5}	3.949177	0.125612	0.9919
1.00×10^{-4}	4.68821524	0.106067	0.9919
1.00×10^{-3}	3.67512488	0.108944	0.9917
1.00×10^{-2}	3.426445	0.117836	0.9915
1.00×10^{-1}	2.6542403	0.116726	0.9785
0.5	1.86405468	0.110053	0.8865
1	1.69960062	0.126101	0.8279
2	1.71188062	0.126135	0.8279
3	1.41218632	0.116203	0.5054
7	1.44990686	0.106237	0.5054
10	1.44930778	0.116169	0.5054
100	1.40948292	0.112271	0.5054
1000	1.42362736	0.111162	0.5054

Table 5: Training Time, t_map, and Test Accuracy for Different Values of Tolerance (LogisticRegression)

Tolerance	Training Time (s)	t_map	Test Accuracy
1.00E-08	1.6301132	0.123822	0.9907
1.00E-06	1.24038302	0.129876	0.9907
1.00E-04	1.22469182	0.106149	0.9907
1.00E-03	0.95976602	0.110314	0.9905
1.00E-02	0.84556846	0.120112	0.9757
2.00E-02	0.76012248	0.118782	0.9164
5.00E-02	0.6875739	0.113143	0.8312
1.00E-01	0.63280088	0.121705	0.5054
1	0.72162804	0.130267	0.5054
10	0.66541596	0.108829	0.5054

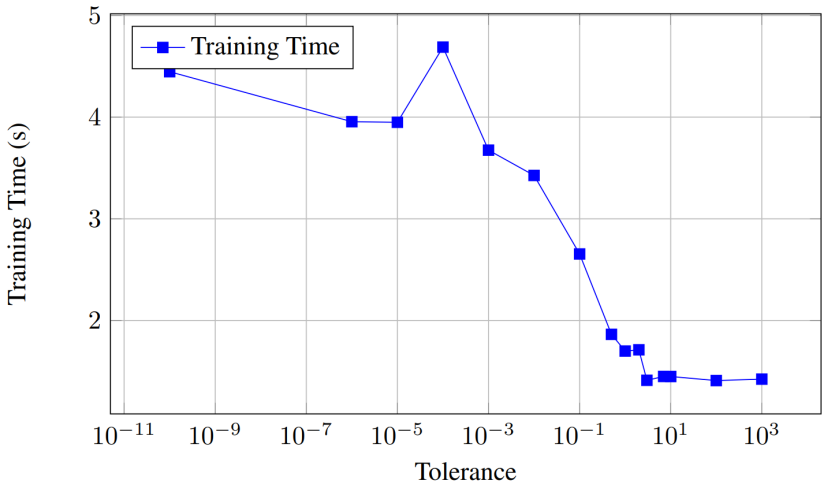

Figure 8: Training Time vs. Tolerance(LinearSVC)

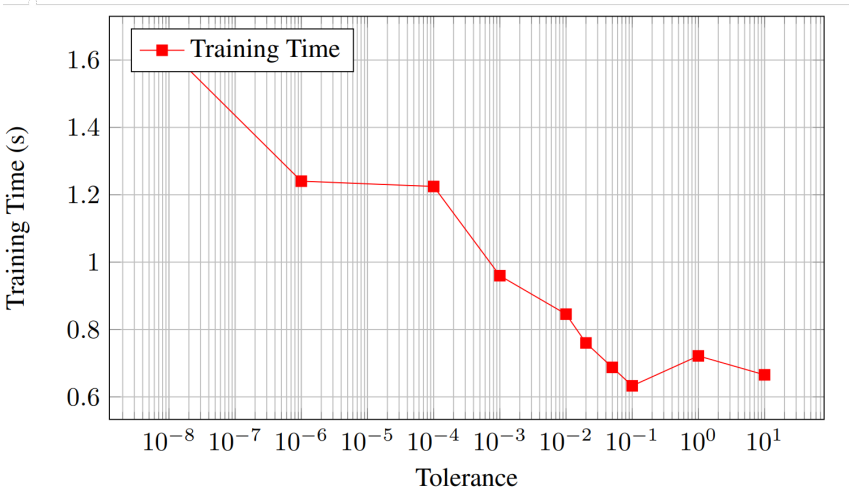

Figure 9: Training Time vs. Tolerance(LogisticRegression)

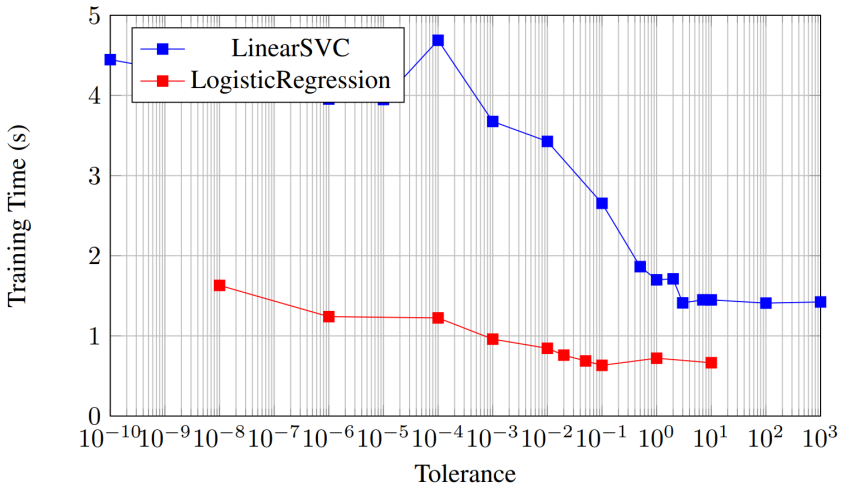

Figure 10: Training Time vs. Tolerance

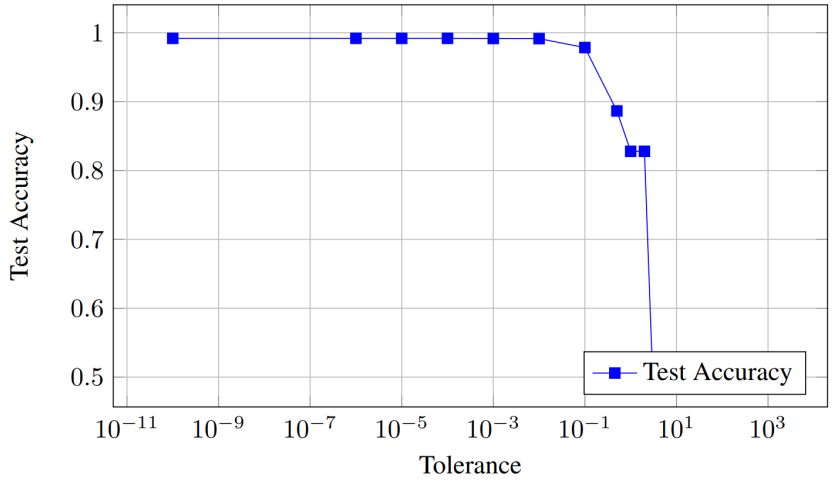

Figure 11: Test Accuracy vs. Tolerance(LinearSVC)

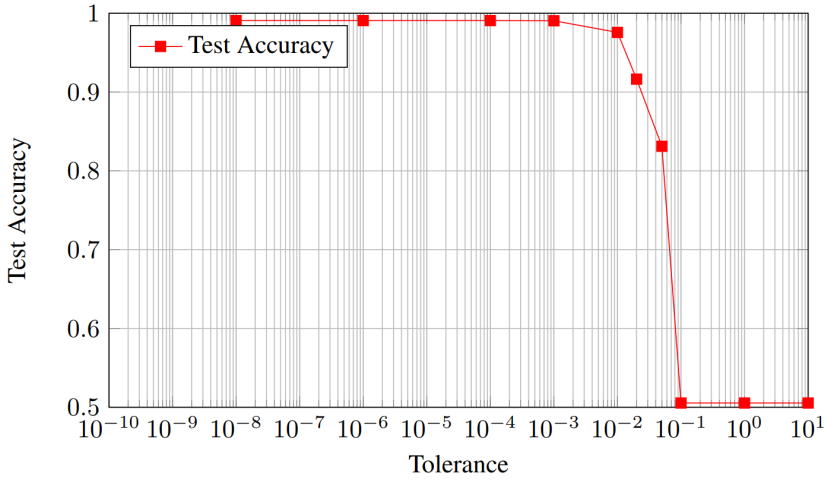

Figure 12: Test Accuracy vs. Tolerance (LogisticRegression)

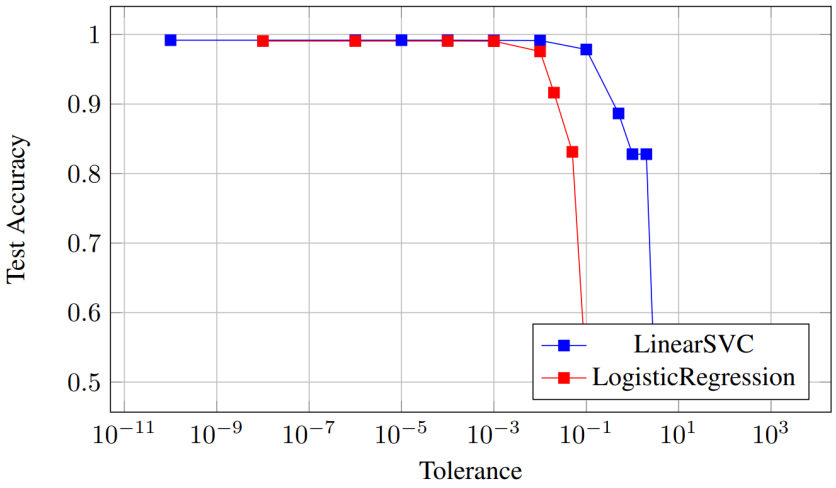

Figure 13: Test Accuracy vs. Tolerance