CS771

Group-Atop

1 Problem 1.1

Uncovering the Mask of Xorro. Melbo was unhappy that the arbiter and XOR PUF devices were broken so easily using a simple linear model. In an effort to make an unbreakable PUF, Melbo came up with another idea devices called ring oscillators. Below, we first describe a traditional ring oscillator, then we describe a more interesting ring oscillator using XOR gates (called a XOR Ring Oscillator or XORRO) and finally we describe how to create a powerful PUF using multiple XORROs.

The simple RO does not allow multiple challengeresponse pairs (CRP) to be created. To remedy this, we notice that the XOR gate can also act as a configurable inverter. The XOR gate takes two inputs and the output is 1 only if exactly one of the inputs is 1. If both or neither of the inputs is 1 then the output is 0. Note that if the second input to a XOR gate is fixed to 1 then the XOR gate starts acting as an inverter with respect to the first input since XOR(0,1) = 1 and XOR(1,1) = 0. Similarly, if the second input to a XOR gate is fixed to 0 then the XOR gate starts acting as an identity with respect to the first input since XOR(0,0) = 0 and XOR(1,0) = 1. Using this, the XOR Ring Oscillator is created as shown in the figure above.

We take two XORROs. Our challenge is an R bit string that is fed into both XORROs as their config bits. Each of the XORROs will now start oscillating. The (oscillating) outputs of the two XORROs is fed into a counter which can find out which XORRO has a higher frequency. If the upper XORRO (in this case XORRO 0) has a higher frequency, the counter outputs 1 else if the lower XORRO (in this case XORRO 1) has a higher frequency, the counter outputs 0. The output of the counter is our response to the challenge. Assume that it will never be the case that both XORROs have the same frequency i.e. the counter will never be confused.

Melbo thinks that with 2^R possible frequencies in each XORRO and 2^S XORROS, there is no way any machine learning model, let alone a linear one, 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 even in this case, there does exist a model that can perfectly predict the responses of a XOR-PUF and this model can be learnt if given enough challenge-response pairs (CRPs).

The following enumerates 4 parts to the question. Three questions will be answered in this report.

1.1 By giving a detailed mathematical derivation (as given in the lecture slides), show how a simple XORRO PUF can be broken by a single linear model. Recall that the simple XORRO PUF has just two XORROs and has no select bits and no multiplexers (see above figure and discussion on Simple XORRO PUF). Thus, the challenge to a simple XORRO PUF has just R bits. More specifically, give derivations for a map $\Phi : (0,1)^R \rightarrow \mathbb{R}^D$ mapping R-bit 0/1-valued challenge vectors to D-dimensional feature vectors (for some D > 0) and show that for any simple XORRO PUF, there exists a linear model i.e. $w \in \mathbb{R}^D, b \in \mathbb{R}$ such that for all challenges $c \in (0,1)^R$, the following expression

$$\frac{1 + sign(w^T \Phi(c) + b)}{2}$$

The output of i_{th} XOR:

$$a_{i+1} = a_i(1 - c_i) + c_i(1 - a_i)$$

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Here c_i is the i_{th} challenge bit and a_i is the other input of the i_{th} XOR.

If initial input a_0 = 0 the $a_1 = c_0$ and $a_2 = c_1 + c_0 - 2c_ic_0$.

If initial input a_1 = 0 the $a_1 = 1 - c_0$ and $a_2 = 1 - c_1 - c_0 - 2c_ic_{c0}$. From noticing the pattern, we can say that, If the output of i_{th} XOR is a_i when initial input is 0, then output if i_{th} XOR is $1 - ai$ when initial input is 1.

The delay associated to ith XOR.

$$f_i(x_i, c_i) = (1 - x_i)(1 - ci)\delta^i_{00} + x_ic_i\delta^i_{11} + (1 - c_i)x_i\delta^i_{10} + (1 - x_i)c_i\delta^i_{01}$$

Here, x_i is input of i_{th} XOR of any particular XORRO and c_i is challenge to the i_{th} XOR of any particular XORRO.

$$Time\,period = \sum_{j=0}^{1}\sum_{i=0}^{R-1} f_i^j(x_i, c_i)$$

$\sum_{i=0}^{R-1} f_i^0(x_i, c_i)$ is the delay corresponding to zero initial input and $\sum_{i=0}^{R-1} f_i^1(x_i, c_i)$ is the delay corresponding to one as initial input.

$$Time\,period\,of\,0_{th}\,XORRO\,T_0 = \sum_{j=0}^{1}\sum_{i=0}^{R-1} f_{i0}^j(x_i, c_i)$$

$$Time\,period\,of\,1_{th}\,XORRO\,T_1 = \sum_{j=0}^{1}\sum_{i=0}^{R-1} f_{i1}^j(x_i, c_i)$$

After applying some simplification and basic calculus

$$T_0 = (c_1 - c_0)(\delta^0_{00} + \delta^0_{10}) + (1 - c_1)(\delta^1_{00} + \delta^1_{10}) + c_1(\delta^1_{11} + \delta^1_{01}) + c_0(\delta^0_{11} + \delta^0_{01})$$

$$T_0 = \sum_{i=0}^{R-1}(1 - c_i)(\delta^i_{00} + \delta^i_{10}) + c_i(\delta_{11i} + \delta^i_{01})$$

$$\Delta = T_1 - T_0$$

$$\Delta = \sum_{i=0}^{R-1}(1 - c_i)((\delta^i_{00} + \delta^i_{10})^1 - (\delta^i_{00} + \delta^i_{10})^0) + c_i((\delta^i_{11} + \delta^i_{01})^1 - (\delta^i_{00} + \delta^i_{10})^0)$$

If $\Delta > 0$ then, $T_1 > T_0$ which implies $f_1 < f_0$ so the output becomes 1.

If $\Delta < 0$ then, $T_1 < T_0$ which implies $f_1 > f_0$ so the output becomes 0.Hence,

$$Output = \frac{1 + sign(\Delta)}{2}$$

1.2 Show how to extend the above linear model to crack an Advanced XORRO PUF. Do this by treating an advanced XORRO PUF as a collection of multiple simple XORRO PUFs. For example, you may use M = $2^{s-1}(2^s - 1)$ linear models, one for each pair of XORROs, to crack the advanced XORRO PUF.

Extending the equations found in part one to answer the problem of **Advanced XORRO PUF**.

$$Time\,period\,of\,p_{th}\,XORRO\,T_p = \sum_{j=0}^{1}\sum_{i=0}^{R-1} f_{ip}^j(x_i, c_i)$$

$$Time\,period\,of\,q_{th}\,XORRO\,T_q = \sum_{j=0}^{1}\sum_{i=0}^{R-1} f_{iq}^j(x_i, c_i)$$

Here p,q $\in$ 0,1,.....,$2^s - 1$. p $\rightarrow$ selected by the upper MUX, q $\rightarrow$ selected by the lower MUX.

$$T_p = \sum_{i=0}^{R-1}(1 - c_i)(\delta^i_{00} + \delta^i_{10}) + c_i(\delta_{11i} + \delta^i_{01})$$

$$\Delta = T_q - T_p$$

$$\Delta = \sum_{i=0}^{R-1} (1 - c_i)((\delta_{00}^i + \delta_{10}^i)^q - (\delta_{00}^i + \delta_{10}^i)^p) + c_i((\delta_{11}^i + \delta_{01}^i)^q - (\delta_{00}^i + \delta_{10}^i)^p)$$

If $\Delta > 0$ then, $T_q > T_p$ which implies $f_q < f_p$ so the output becomes 1.

If $\Delta < 0$ then, $T_q < T_p$ which implies $f_q > f_p$ so the output becomes 0.Hence,

$$Output = \frac{1 + sign(\Delta)}{2}$$

Total number of models = $\binom{16}{2}$ = 120.

1.3 Report outcomes of experiments with both the sklearn.svm.LinearSVC and sklearn.linear model.LogisticRegression methods when used to learn the ensemble linear model. In particular, report how various hyperparameters affected training time and test accuracy using tables, 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 the affect of training time and test accuracy of at least 2 of the following

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

For the Hinge Loss Function :
We have varied the **C hyperparameter** from 1-500 by taking a step length of 10.And by doing that we have been able to achieved max accuracy of 93.96 with average training time of 5.435 sec.

For the Squared Hinge Loss Function:
Here also we have varied the **C hyperparameter** from 1-500 by taking a step length of 10.And by doing so we have get an accuracy of 94.74 with average training time of 5.4621 sec.
From the above results we can conclude that accuracy for the square hinge loss function is more than that of hinge loss function but the average training time for the hinge loss function is less.

b) setting C hyperparameter in LinearSVC and LogisticRegression to high/low/medium values

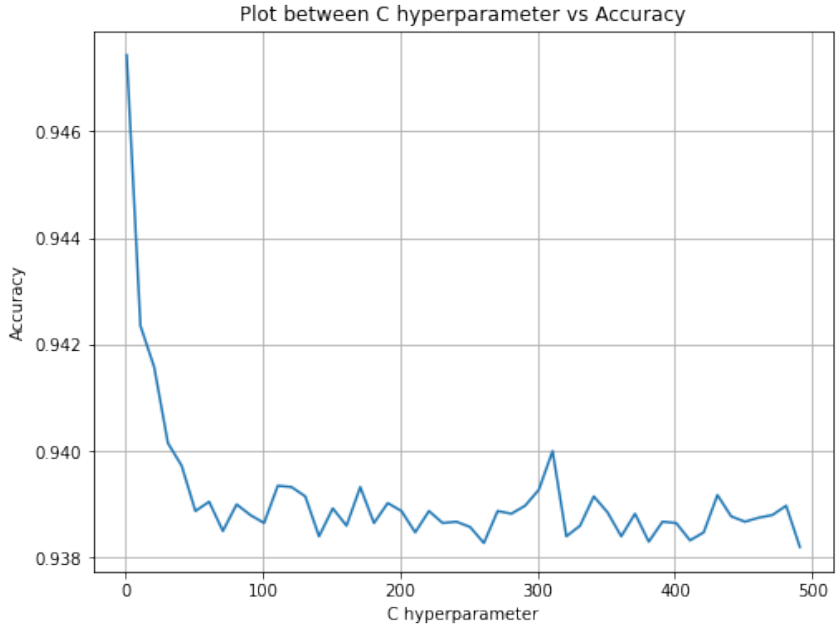

Figure 1: LinearSVC Accuracy vs c

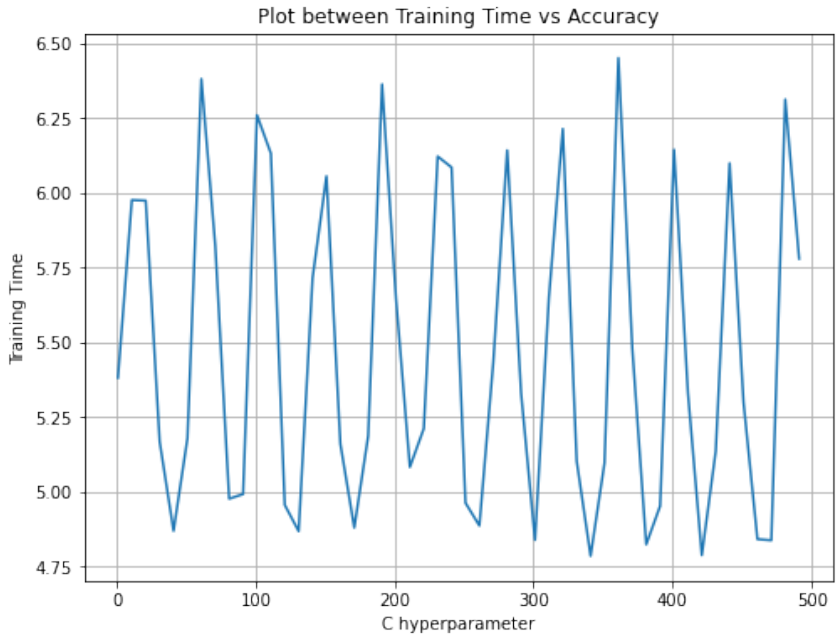

Figure 2: LinearSVC Training Time vs c

The following two plots are drawn for logisticregression having penalty of **l2** and solver **lbfgs**. The average training time =9.0949532 sec and maximum accuracy =0.9499 for C=21.

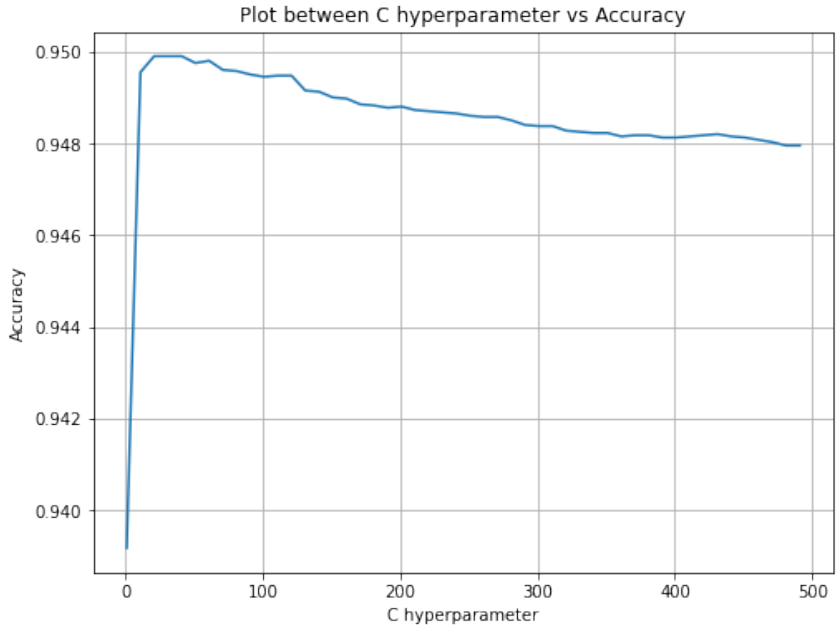

Figure 3: LogisticRegression Accuracy vs c

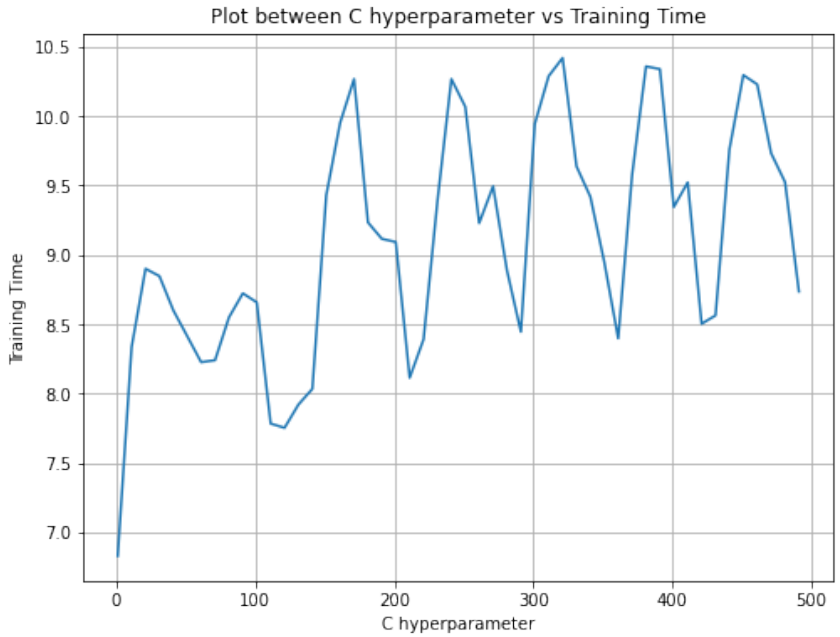

Figure 4: LogisticRegression Training Time vs c

The following two plots are drawn for logisticregression having penalty of **l2** and solver **liblinear**. The average training time =4.37810 sec and maximum accuracy =0.9495 .

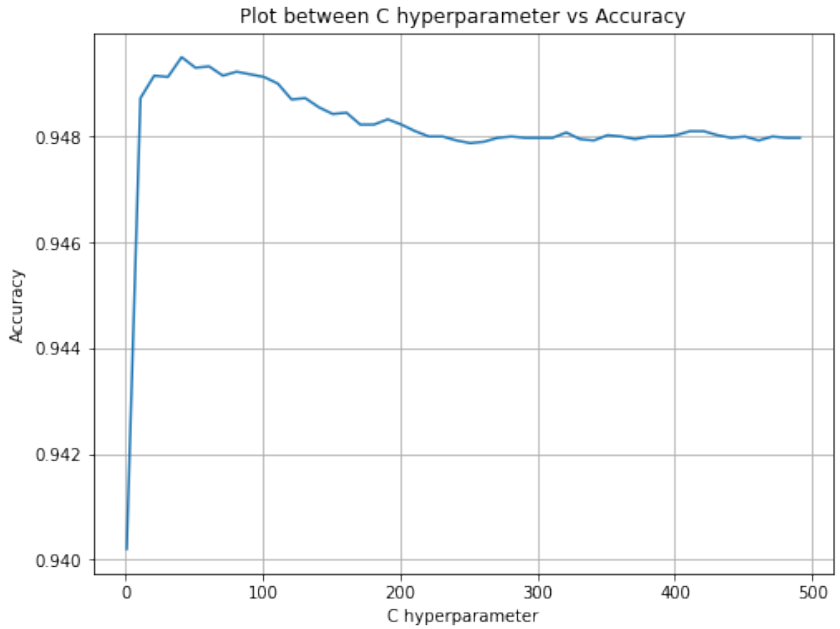

Figure 5: LogisticRegression liblinear Accuracy vs c

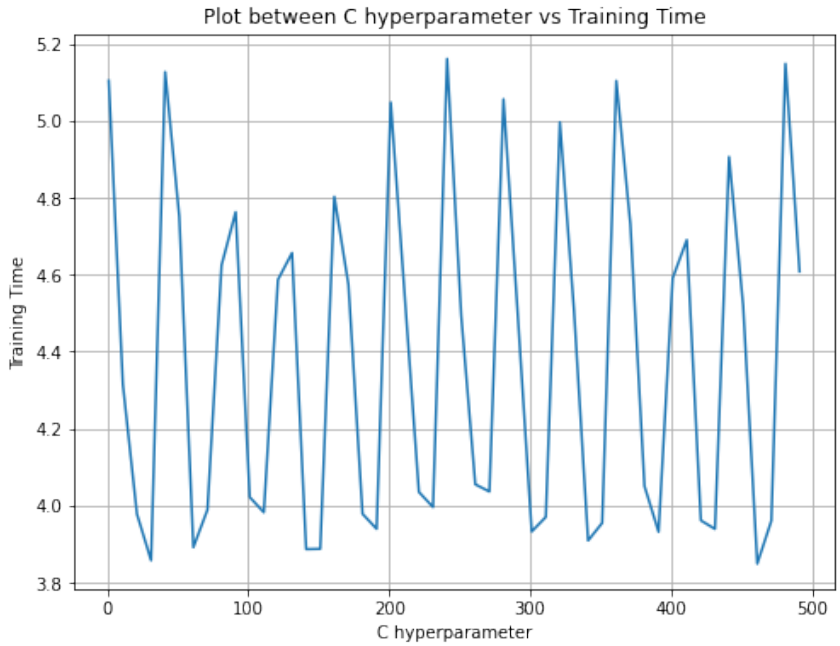

Figure 6: LogisticRegression liblinear Training Time vs c for

The following two plots are drawn for logisticregression having penalty of **l1** and solver **liblinear**. The average training time =4.62921 sec and maximum accuracy =0.9483 .

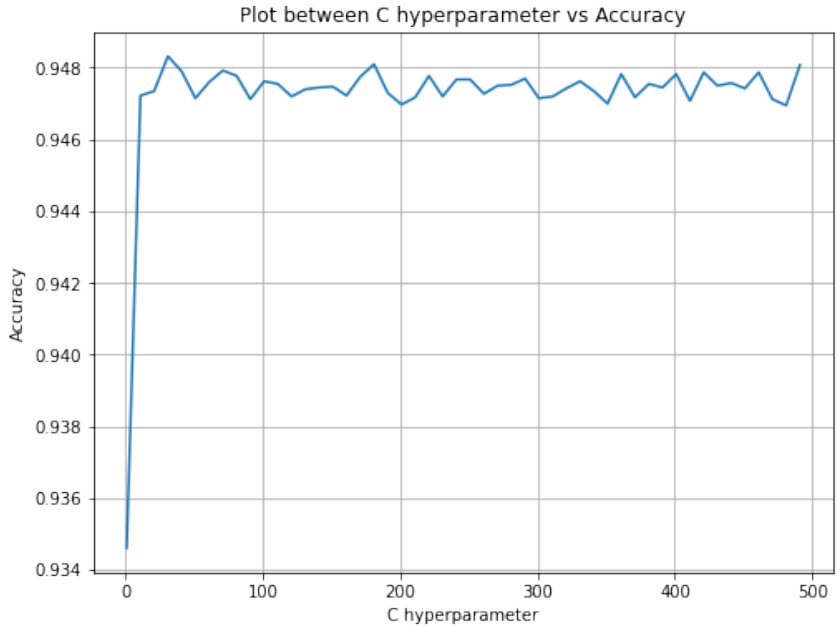

Figure 7: Logisticregression liblinear Accuracy vs c

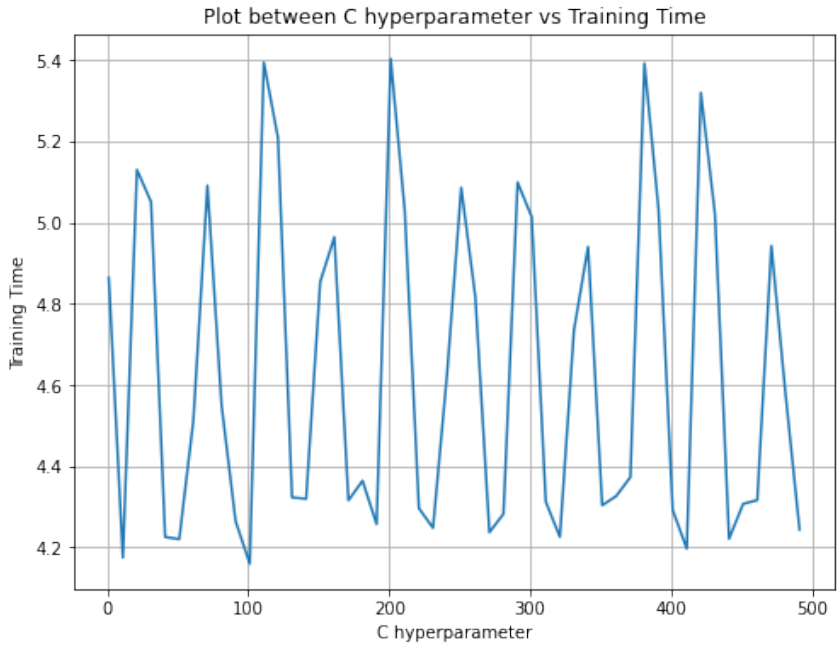

Figure 8: LogisticRegression liblinear Training Time vs c

c)changing the tol hyperparameter in LinearSVC and LogisticRegression to high/low/medium values

LinearSVC(loss='squarehinge', C=1)

Tolerance Value	Training Time	Accuracy
0.1	4.5794	0.94732
0.01	5.3193	0.94741
0.001	5.7411	0.94743
0.0001	6.0504	0.94736
0.00001	6.0360	0.94737
0.000001	6.3778	0.947335

From the above results we can conclude that maximum accuracy is obtained with minimum training time when C=1 and tolerance=0.001 for LinearSVC .

LogisticRegression(C=21 , solver='lbfgs', penalty='l2')

Tolerance Value	Training Time	Accuracy
0.1	5.5367	0.9495
0.01	5.8368	0.949925
0.001	6.7326	0.949925
0.0001	7.8749	0.9498
0.00001	8.653	0.9498
0.000001	8.0669	0.9498

From the above results we can conclude that maximum accuracy is obtained with minimum training time when C=21 and tolerance=0.01 for logistic regression .

From the above we can observe that training time is inversely propotional to tolerance value for both LogisticRegression and LinearSVC case.

d) changing the penalty (regularization) hyperparameter in LinearSVC and LogisticRegression (l2 vs l1)
For LogisticRegression(C=21, solver='liblinear') accuracy decreases from 0.9495 on l2 to 0.948325 on l1 and training time increases from 4.3781 sec on l2 to 4.6292 sec on l1.

For LinearSVC(loss='squarehinge', C=1) accuracy decreases from 0.94743 on l2 to 0.946105 on l1 and training time increases from 5.7411 sec on l2 to 10.2994 sec on l1.