CS771 Assignment 1 Report

Sankalp Mittal
220963
sankalpm22@cse.iitk.ac.in

Animesh Madaan
220145
manimesh@iitk.ac.in

Akanksha Wattamwar
221214
akankshab22@iitk.ac.in

Tanmay Siddharth
221129
tanmays22@iitk.ac.in

Raghav Govind
220852
graghav22@iitk.ac.in

Abstract

This report is written for Assignment 1 of the CS771 2023-2024 Even semester course offering. The question was related to finding out the best linear model for hacking the encryption offered by **Companion ArbiteR PUF (CAR-PUF's)**.

1 Finding out the feature vector map

Derivation

We aim to show how a single linear model can break a CAR-PUF. Let $\phi : \{0,1\}^{32} \to \mathbb{R}^D$ be a map that maps 32-bit $0/1$-valued challenge vectors to D-dimensional feature vectors, where $D > 0$.

For any CAR-PUF, there exists a D-dimensional linear model $W \in \mathbb{R}^D$ 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:

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

Proof

Problem Statement: Let $(\mathbf{u}, p)$ and $(\mathbf{v}, q)$ be the two linear models that can exactly predict the outputs of the two arbiter PUFs sitting inside the CAR-PUF. W and b will depend on u, v, p, q, τ. Note that $\phi(c)$ must depend only on c (and perhaps universal constants such as 2, $\sqrt{2}$, etc.). The map ϕ must not use PUF-specific constants such as u, v, p, q, τ.

Proof Strategy:

1. Derive expressions for the linear models $(\mathbf{u}, p)$ and $(\mathbf{v}, q)$ predicting the outputs of the two arbiter PUFs.
2. Establish a relationship between the outputs of the CAR-PUF and the linear models.
3. Show that there exists a D-dimensional linear model W and a bias term b satisfying Equation (1) for all CRPs (c, r), and give its explicit solution.

Step 1: The Linear Models (u, p) and (v, q):

As solved in the discussion hours, the time delay of the two working and reference PUFs is

$$\Delta_w = \mathbf{u}^\top \cdot \mathbf{x} + p$$

$$\Delta_r = \mathbf{v}^\top \cdot \mathbf{x} + q$$

where $\mathbf{x_i}$ = $\prod_{n=i}^{32} d_i$, and $d_i = 1 - 2c_i$, where c_i are the challenges.

Step 2: Establishing Relationship with CAR-PUF:

Now, $|\Delta_w - \Delta_r| > \tau \iff (\Delta_w - \Delta_r - \tau) \cdot (\Delta_w - \Delta_r + \tau) > 0$.

Expanding the terms, we get

$$(\Delta_w - \Delta_r)^2 - \tau^2 > 0$$
$$\implies ((\mathbf{u}^\top - \mathbf{v}^\top) \cdot \mathbf{x} + (p-q))^2 - \tau^2 > 0$$
$$\implies (\mathbf{u}^\top - \mathbf{v}^\top) \cdot \mathbf{x}\mathbf{x}^\top \cdot (\mathbf{u} - \mathbf{v}) + 2(p-q)(\mathbf{u}^\top - \mathbf{v}^\top) \cdot \mathbf{x} + (p-q)^2 - \tau^2 > 0$$

This is a linear function in $x_i x_j$ and $x_i \quad \forall i, j \in \{1, 2, \ldots, 32\}$, and as $x_i \in \{-1, 1\} \implies x_i^2 = 1 \quad \forall i, j \in \{1, 2, \ldots, 32\}$, it is a linear function of $x_i x_j$ and $x_i \quad \forall i \neq j \in \{1, 2, \ldots, 32\}$.

Step 3: Deriving the Linear Model for the CAR-PUF:

Therefore, the CAR-PUF can indeed be broken by a single linear model, as demonstrated by the feature map as follows:

$\phi : \{0,1\}^{32} \rightarrow \mathbb{R}^{528}, \quad \phi(c) = (x_1x_2, x_1x_3, \ldots, x_{31}x_{32}, x_1, \ldots, x_{32})^\top$,

The response to a query $\mathbf{c}$ is

$$\frac{1 + \text{sign}(W^\top \phi(c) + b)}{2} \tag{2}$$

2 Experimentation with different Linear Classifiers

We have trained various models to get the best test accuracy and least training time. We could achieve the best accuracy of 99.31% with a training time of around 5 seconds with Logistic Regression.

Below is the report of all the experiments we performed with various models.

2.1 Logistic Regression

2.1.1 Background

Scikit-learn's logistic regression is a simple yet effective algorithm for binary classification tasks. It fits a logistic curve to the data, allowing it to predict the probability of an instance belonging to a particular class. It supports regularization, multiple optimization solvers, provides probability estimates, and is efficient for large datasets.

The default values of the parameters of the model are:

- **solver:** 'lbfgs'
- **C:** 100
- **tol:** 0.00001

2.1.2 Experiments

Changing Solver: We have used the *liblinear*, *lbfgs*, *sag* solvers.

Setting the rest of the features to their default values, the following results were obtained.

Table 1: Changing Solvers

Solver	Train Accuracy	Test Accuracy	Train Time
liblinear	99.9975%	99.31%	20.353s
lbfgs	100.0%	99.31%	5.391s
sag	99.8025%	99.19%	40.609s

Clearly, **'lbfgs'** is the best solver as it gives higher accuracy with significantly lower times.

Changing C (Regularization): We will test for various C values leaving the other values as default.

Table 2: Changing Regularization

Penalty	Train Accuracy	Test Accuracy	Train Time
10	99.8925%	99.21%	5.99s
100	100.0%	99.31%	5.347s
1000	100.00%	99.25%	5.88s

Clearly, C **= 100** gives better results with respect to accuracy.

Changing Tolerance Value: Since the default value is 0.0001, we will try out values lesser and greater than this to see the impact, leaving the rest of the parameters as the default values.

Table 3: Changing Tolerance Values

Tolerance	Train Accuracy	Test Accuracy	Train Time
1e-3	99.895%	99.119%	5.90s
1e-4	99.997%	99.27%	6.220s
1e-5	100.0%	99.31%	5.356s
1e-6	100.0%	99.31%	6.816s
1e-7	100.0%	99.31%	10.231s

Here, **larger** values for tolerance are marginally better. Since train time increases, $tol = 0.00001$ is the optimum value.

2.2 SGD Classifier

2.2.1 Background

This is a part of the **sklearn.linear_model** library and it implements linear classifiers (SVM, logistic regression, etc.) with **Stochastic Gradient Descent (SGD)** training.

This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning via the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM).

The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared Euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net).

The default values of the parameters of the model are:

- **loss:** "hinge"
- **penalty:** "l2"
- **alpha:** 0.0001
- **tol:** 0.001

2.2.2 Experiments

Changing Loss Function: We have used the *hinge*, *modified_huber*, *squared_hinge* loss functions.

Setting the rest of the features to their default values, the following results were obtained.

Table 4: Changing Loss Functions

Loss Function	Train Accuracy	Test Accuracy	Train Time
hinge	98.0125%	97.78%	5.757s
squared_hinge	99.655%	98.94%	7.861s
modified_huber	98.51%	98.88%	8.379s

Clearly, **"hinge"** is the best as it gives much lower times, and the compromise on accuracy is not too much.

Changing Penalty: We will test for *l2* and *l1* norm, leaving the other values as default.

Table 5: Changing Penalty type

tol	Train Accuracy	Test Accuracy	Train Time
l1	99.97%	99.08%	6.238s
l2	98.0125%	97.78%	5.757s

Clearly, **l1 norm** gives extremely better results than l2 norm, with respect to accuracy and time compromise is not that much, so that should clearly be the preferred choice.

Changing Tolerance Value: Since the default value is 0.001, we will try out values lesser and greater than this to see the impact, leaving the rest of the parameters as the default values.

Table 6: Changing Tolerance Values for l2 penalty

tol	Train Accuracy	Test Accuracy	Train Time
1e-6	98.6275%	98.16%	7.325s
1e-3	98.0125%	97.78%	5.757s
1e-1	96.41%	96.345%	3.938s

Clearly, **medium** values for *tol* are better as there is a good balance between the accuracy and time taken

We will also try using the l1 penalty as that gives much better results

Table 7: Changing Tolerance Values for l1 penalty

tol	Train Accuracy	Test Accuracy	Train Time
1e-6	99.965%	99.04%	6.979s
1e-3	99.97%	99.08%	6.238s
1e-1	99.925%	99.1%	4.343s

Here Clearly **larger** values for *tol* are better, although marginally, as not a lot of changes are observed.

2.3 LinearSVC

2.3.1 Background

LinearSVC, or Linear Support Vector Classifier, is a variant of Support Vector Machine (SVM) used for classification tasks. SVM is a powerful supervised learning algorithm widely used for classification and regression tasks. LinearSVC specifically focuses on linear classification problems, where it separates classes using a hyperplane.

2.3.2 Experiments

The default values of the parameters of the model are:

- **C:** 1
- **tol:** 0.00001
- **penalty:** l2
- **loss:** squared_hinge

Changing Loss function: We have used the *squared_hinge*, *hinge*.

Setting the rest of the features to their default values, the following results were obtained.

Table 8: Changing Loss Function

Loss function	Test Accuracy	Train Time
squared_hinge	99.11%	16.93s
hinge	98.87%	16.18s

'Squared_hinge' gives higher accuracy compared to **'hinge'**.

Changing C (Regularization): We will test for various C values leaving the other values as default.

Clearly, C **= 1** gives better results with respect to accuracy.

Changing Tolerance Value: Since the default value is 0.0001, we will try out values lesser and greater than this to see the impact, leaving the rest of the parameters as the default values.

Here, **medium** values for tolerance are marginally better. Since train time increases, $tol = 0.00001$ is the optimum value.

Table 9: Changing Regularization

Penalty	Test Accuracy	Train Time
1	99.11%	16.93s
10	99.06%	16.63s
100	98.99%	16.28s
1000	98.93%	17.176s

Table 10: Changing Tolerance Values

Tolerance	Test Accuracy	Train Time
1e-3	98.93%	17.613s
1e-4	98.90%	16.35s
1e-5	99.11%	16.93s
1e-6	98.82%	16.29s
1e-7	99.05%	18.55s

2.4 Ridge Classifier

2.4.1 Background

The **linear_model** module in the **scikit-learn** library provides implementations of various linear classifiers and regressors. One of the classifiers available is the Ridge classifier, which belongs to the family of regularized linear models.

The Ridge classifier implements regularized linear models using the Ridge regression algorithm. This algorithm uses the L2 regularization technique, which adds a penalty term to the loss function to control overfitting by shrinking the model parameters towards zero.

The Ridge classifier in scikit-learn works with data represented as dense or sparse arrays of floating-point values for the features. It is capable of handling large datasets and supports online learning via the partial_fit method, making it suitable for scenarios where data arrives in batches or when memory constraints exist.

One of the key features of the Ridge classifier is its ability to handle multicollinearity in the input features, which can often lead to unstable parameter estimates in ordinary least squares regression. By adding the L2 regularization term, Ridge regression mitigates this issue by imposing a penalty on large coefficients, effectively reducing their impact on the model's predictions.

The default values of the parameters of the model are:

- **alpha:** 1
- **tol:** 0.001
- **max_iter:** 1000
- **solver:** "auto"

2.4.2 Experiments

Changing alpha: We tried changing the values of *alpha* with rest of the parameters as default and the results are as follows:

Table 11: Changing alpha value

Value of alpha	Train Accuracy	Test Accuracy	Train Time
1	83.857%	82.91%	7.355s
100	83.855%	82.91%	6.423s
800	83.835%	82.91%	7.490s
900	83.832%	82.90%	10.035s
1000	83.832%	82.89%	6.107s

The best accuracy we could achieve with this model is 82.91% which we get when alpha is between 1 to 800 as its value further increases, the accuracy decreases.

Changing Tolerance Value: Since the default value is 0.001, we will try out values lesser and greater than this to see the impact, leaving the rest of the parameters as the default values.

Table 12: Changing Tolerance Values

tol	Train Accuracy	Test Accuracy	Train Time
1e-6	83.857%	82.91%	9.140s
1e-4	83.857%	82.91%	6.224s
1e-3	83.857%	82.91%	6.29s
1e-1	83.857%	82.91%	6.685s

Clearly, **medium** values for *tol* are better as there is a good balance between the accuracy and time taken.

Changing Solver: We will test for some other values of solver other than **auto** leaving the other values as default.

Table 13: Changing Solver type

Solver	Train Accuracy	Test Accuracy	Train Time
auto	83.857%	82.91%	6.29s
svd	83.857%	82.91%	12.75s
cholesky	83.857%	82.91%	6.355s

Clearly, **auto solver** is good compared to other solvers due to less time taken to train the model.