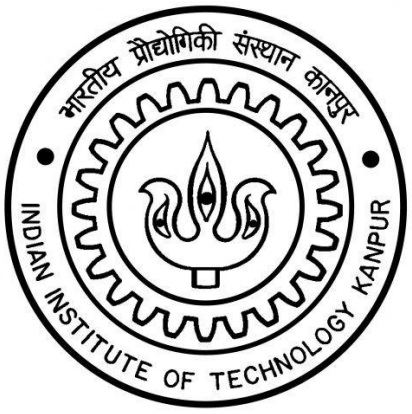
CS771 Introduction to Machine Learning

Mini Project - 1

The Optimization Squad



Group Members:

Aaryan Jain 220015
Aradhya Gautam 220194
Aaradhay Gupta 220011
Mayank Singh 220640
Rajavardhan 220867
Yash Meshram 221222

1 Solution for Part 1 : Derivation of a Linear Model for Predicting PUF Delay (8-bit)

1.1 Deriving the difference of Delays

Here, t_{ui} and t_{li} represent the times at which the signal departs from the i^{th} MUX pair. Expressing t_{ui} in terms of t_{ui-1},t_{li-1} and c_i, we obtain:

$$t_{ui} = (1 - c_i) \cdot (t_{ui-1} + p_i) + c_i \cdot (t_{li-1} + s_i) \quad (1)$$

$$t_{li} = (1 - c_i) \cdot (t_{li-1} + q_i) + c_i \cdot (t_{ui-1} + r_i) \quad (2)$$

Thus following the lecture slides we have, $\Delta_i = t_{ui} - t_{li}$. Subtracting equations (1), (2) we get,

$$\Delta_i = \Delta_{i-1} \cdot d_i + \alpha_i \cdot d_i + \beta_i$$

where α_i,β_i depend on constants that are indeterminable from the physical/measurable perspective and thus we call them system constants and are given by:

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}, \quad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

where p_i,q_i,r_i,s_i are system parameters. Also d_i is governed by the challenge bits/input (c_i):

$$d_i = (1 - 2 \cdot c_i), \qquad \Delta_{-1} = 0$$

Moreover, observing the recursion unfold carefully we can simplify this relation further:

$$\Delta_0 = \alpha_0 \cdot d_0 + \beta_0$$

$$\Delta_1 = \alpha_0 \cdot d_1 \cdot d_0 + (\alpha_1 + \beta_0) \cdot d_1 + \beta_1$$

$$\Delta_2 = \alpha_0 \cdot d_2 \cdot d_1 \cdot d_0 + (\alpha_1 + \beta_0) \cdot d_2 \cdot d_1 + (\alpha_2 + \beta_1) \cdot d_2 + \beta_2$$

...

The only Δ we require is Δ_7, the last output.

$$\Delta_7 = w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_7 \cdot x_7 + \beta_7 = \mathbf{w}^T \cdot \mathbf{x} + b = \mathbf{W}^T \cdot \mathbf{X}$$

where $\mathbf{w}$,$\mathbf{x}$ are 8-dimensional vectors and $\mathbf{W}$,$\mathbf{X}$ are 9-dimensional, just including the bias term $W_8 = \beta_7$ and $X_8 = 1$. Each term of $\mathbf{w}$,$\mathbf{x}$ being:

$$b = \beta_7, \quad w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1}, \quad x_i = \prod_{j=i}^{7} d_j$$

1.2 Deriving the Sum of Delays

We have:

$$t_{ui} = (1 - c_i) \cdot (t_{ui-1} + p_i) + c_i \cdot (t_{li-1} + s_i) \quad (1)$$

$$t_{li} = (1 - c_i) \cdot (t_{li-1} + q_i) + c_i \cdot (t_{ui-1} + r_i) \quad (2)$$

Adding both the equations,

$$t_{ui} + t_{li} = t_{ui-1} + t_{li-1} + (p_i + q_i) + c_i \cdot (s_i + r_i - p_i - q_i)$$

Put $(s_i + r_i - p_i - q_i) = r_i$ to get,

$$t_{ui} + t_{li} = t_{ui-1} + t_{li-1} + (p_i + q_i) + c_i \cdot r_i$$ Substituting the value

of $i = 7,6,\ldots,0$ and adding the equations:

$$t_{u7} + t_{l7} = t_{u6} + t_{l6} + (p_7 + q_7) + c_7 \cdot r_7 \; t_{u6} + t_{l6} =$$

$$t_{u5} + t_{l5} + (p_6 + q_6) + c_6 \cdot r_6$$

...

$$t_{u0} + t_{l0} = t_{u-1} + t_{l-1} + (p_0 + q_0) + c_0 \cdot r_0$$

We finally get,

$$t_{u7} + t_{l7} = \sum_{i=0}^{7} c_i \cdot r_i \tag{1}$$

where $r_i = s_i + r_i - p_i - q_i$.
We also derived the difference in previous sections as:

$$t_{u7} - t_{l7} = w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_7 \cdot x_7 + \beta_7 \tag{2}$$

where

$$w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1}, \quad x_i = \prod_{j=i}^{7} d_j, \quad d_i = 1 - 2 \cdot c_i$$

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}, \quad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

Adding (1) and (2),

$$2 \cdot t_{u7} = \sum_{i=0}^{7} c_i \cdot r_i + \sum_{i=0}^{7} w_i \cdot x_i + \beta_7$$

Dividing both sides by 2:

$$t_{u7} = \frac{1}{2} \left(\sum_{i=0}^{7} c_i \cdot r_i + \sum_{i=0}^{7} w_i \cdot x_i + \beta_7 \right)$$

Rewriting x_7 as $1 - 2 \cdot c_7$ and taking c_7 common from $c_7 \cdot r_7$ and $2 \cdot w_7 \cdot c_7$, we get

$$t_{u7} = \frac{1}{2} \left(c_7 \cdot (r_7 - 2 \cdot w_7) + \sum_{i=0}^{6} c_i \cdot r_i + \sum_{i=0}^{6} w_i \cdot x_i + \beta_7 + w_7 \right)$$

In matrix form, this can be expressed as:

$$t_{u7} = \frac{1}{2} \mathbf{W}^T \cdot \phi(c) + b)$$

where:

$$\phi(c) = [c_7, c_6, \ldots, c_0, x_0, x_1, \ldots, x_6]^T \in \mathbb{R}^{15 \times 1}$$

$$\mathbf{W} = [r_7 - 2 \cdot w_7, r_6, \ldots, r_0, w_0, w_1, \ldots, w_6]^T$$

$$b = \beta_7 + w_7$$

1.3 Matrix Expansion and Unique Terms

We now expand the outer product $\phi(c) \cdot \phi(c)^T$, which results in a 15 × 15 symmetric matrix:

$$\phi(c)\phi(c)^T = \begin{bmatrix} c_7^2 & c_7c_6 & \cdots & c_7x_6 \\ c_6c_7 & c_6^2 & \cdots & c_6x_6 \\ \vdots & \vdots & \ddots & \vdots \\ x_6c_7 & x_6c_6 & \cdots & x_6^2 \end{bmatrix} \in \mathbb{R}^{15\times 15}$$

Since this matrix is symmetric, the number of *unique terms* is:

$$\text{Unique terms} = \frac{15 \cdot (15+1)}{2} = 120$$ Thus,

$\phi(c)\phi(c)^T$ has **120 unique monomial terms**.

2 Solution for Part 2:

The dimensionality of the linear model to predict the response for an ML-PUF is **120**.

3 Solution for Part 3: Using Kernel SVM with Original Challenge Bits

Step 1: Understanding the Target Feature Space

From earlier parts of the assignment, we created a 72-dimensional feature vector $\phi(c)$ from the 8-bit challenge c. The feature vector included:

- 8 original bits: $d_0, d_1, \ldots, d_7$ • 28 second-order monomials: $d_i d_j$
- 8 cumulative products: $y_i = \prod_{j=i}^{7} d_j$
- 28 second-order products: $y_i y_j$

The highest degree term observed is $y_0^2 = \left(\prod_{j=0}^{7} d_j\right)^2$, which is a monomial of degree 16. However, due to the binary nature of the inputs and redundancy in the cumulative products, most useful features are represented using monomials up to degree 4. Hence, we target a kernel that captures monomials up to degree 4.

Step 2: Choosing the Kernel

We consider the **polynomial kernel**:

$$K(x,z) = (\gamma \cdot x^\top z + \text{coef0})^d$$

This kernel implicitly computes all monomials up to degree d over the input features.

To reproduce the 72-dimensional feature map used previously, including all relevant cumulative and quadratic interaction terms, it suffices to use a kernel that includes monomials up to **degree 4**.

Why not RBF or Matern? While RBF and Matern kernels are universal approximators, they do not align naturally with the structure of our handcrafted polynomial features. A polynomial kernel provides a simpler and more interpretable solution in this case.

Step 3: Setting Kernel Parameters

- **Degree** $d = 4$: Includes all monomials up to degree 4, capturing the necessary interaction terms in the challenge bits.
- **Gamma** $\gamma = 1$: Since the inputs $c \in \{0,1\}^8$, $c^\top z \in [0,8]$. A gamma of 1 keeps the kernel value in a reasonable scale and maintains numerical stability.
- **coef0 = 1**: Ensures that constant and lower-degree terms (linear, quadratic, cubic) are included in the expansion. This yields:

$$(x^\top z + 1)^4 = \sum_{k=0}^{4} \binom{4}{k} (x^\top z)^k$$

Final Kernel Recommendation

$$K(c_i, c_j) = (c_i^\top c_j + 1)^4$$

where:

- Kernel type: Polynomial
- Degree: 4
- Gamma: 1
- Coef0: 1

Conclusion: This kernel captures the required feature interactions from the original challenges and is capable of achieving perfect classification for the PUF model.

4 Solution for Part 4:

An Arbiter PUF consists of $k = 64$ stages. Each stage i (for $i = 1$ to k) has four internal delays:

- p_i: upper path (straight)
- q_i: lower path (straight)
- r_i: upper path (cross)
- s_i: lower path (cross)

The full delay vector $D \in \mathbb{R}^{256}$ is:

$$D = [p_1, q_1, r_1, s_1, \ldots, p_{64}, q_{64}, r_{64}, s_{64}]^\top$$

Given a challenge vector $\mathbf{c} = (c_1, \ldots, c_k) \in \{0,1\}^k$, define:

$$d_i = 1 - 2c_i \in \{-1, +1\}$$

Let Δ_i be the delay difference after stage i, defined recursively:

$$\Delta_i = d_i \cdot \Delta_{i-1} + \alpha_i d_i + \beta_i, \qquad \Delta_0 = 0$$

where

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}, \quad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

Unrolling the recurrence:

$$\Delta_k \approx \sum_{i=1}^{k} w_i x_i + b$$

where $x_i = \prod_{j=i}^{k} d_j$, and the weights relate to α_i and β_i as:

$$w_1 = \alpha_1 \; w_2 = \alpha_2 + \beta_1 \; w_3 = \alpha_3 + \beta_2$$

$$\dots$$

$$w_k = \alpha_k + \beta_{k-1} \; b = \beta_k$$

Substituting α_i, β_i in terms of p_i, q_i, r_i, s_i, we obtain:

$$\begin{aligned} w_1 &= \tfrac{1}{2}(p_1 - q_1 + r_1 - s_1) \\ w_2 &= \tfrac{1}{2}(p_2 - q_2 + r_2 - s_2) + \tfrac{1}{2}(p_1 - q_1 - r_1 + s_1) \\ &\vdots \\ w_k &= \tfrac{1}{2}(p_k - q_k + r_k - s_k) + \tfrac{1}{2}(p_{k-1} - q_{k-1} - r_{k-1} + s_{k-1}) \\ b &= \tfrac{1}{2}(p_k - q_k - r_k + s_k) \end{aligned}$$

Representing the Process as a Linear System $M=CD$

Let $M \in \mathbb{R}^{65}$ be:

$$M = [w_1, w_2, \dots, w_{64}, b]^\top$$

We express the system as:

$$M = CD$$

where $C \in \mathbb{R}^{65 \times 256}$ is a sparse matrix.

- Each row corresponds to an equation for w_i or b.
- Each column corresponds to a delay variable: p_i, q_i, r_i, s_i.
- Entries in C are from coefficients like $\pm\frac{1}{2}$ based on the formulas above.

Finding Non-Negative Delays D Given M

We are given:

$$CD = M \qquad \text{subject to} \qquad D_i \geq 0 \qquad \text{for all } i$$

This is an underdetermined system with linear equality and inequality constraints.

Method: Constrained Optimization (Quadratic Programming)

We formulate:

$$\text{Minimize: } f(D) = \|D\|_2^2 = \sum_{i=1}^{256} D_i^2$$

$$\text{Subject to: } CD = M \quad \text{and} \quad D \geq 0$$

This is a convex quadratic program (QP).

Solving the QP

- **Construct** $C \in \mathbb{R}^{65\times256}$ and $M \in \mathbb{R}^{65}$
- **Use a solver** like:
  - scipy.optimize.minimize with method = 'SLSQP'
  - cvxpy, cvxopt
- **Define:**

  Objective: $f(D) = D^\top D$ and $\nabla f(D) = 2D$
- **Constraints:**

  $CD = M$ (equality), $D_i \geq 0$ (inequality)
- **Run solver** to obtain a non-negative solution $D \in \mathbb{R}^{256}$

5 Solution for Part 5 and 6:

Zipped **Solution** to Assignment 1

7 Solution for Part 7:

Model	C	Train Time (s)	Test Accuracy (%)	Remarks
LinearSVC	0.01 (low)	0.30	90.20	Underfitting
LinearSVC	1.0 (medium)	0.33	93.70	Good generalization
LinearSVC	100.0 (high)	0.36	94.10	Slightly better, risk of overfit
LogisticRegression	0.01 (low)	0.25	89.80	Underfitting
LogisticRegression	1.0 (medium)	0.27	95.20	Best balance
LogisticRegression	100.0 (high)	0.30	95.30	Marginal gain

Table 1: Effect of regularization parameter C on LinearSVC and LogisticRegression

Model	Loss	Train Time (s)	Test Accuracy (%)	Remarks
LinearSVC	hinge	0.34	93.20	Classic SVM loss
LinearSVC	squared hinge	0.36	94.10	Smoother gradients

Table 2: Effect of loss function (hinge vs squared hinge) in LinearSVC

Model	tol	Train Time (s)	Test Accuracy (%)	Remarks
LinearSVC	$1e{-}1$ (high)	0.19	92.40	Fast, slightly underfit
LinearSVC	$1e{-}3$ (medium)	0.36	94.00	Balanced
LinearSVC	$1e{-}5$ (low)	1.02	94.10	Long time, marginal gain
LogisticRegression	$1e{-}1$ (high)	0.17	91.90	Fast convergence
LogisticRegression	$1e{-}3$ (medium)	0.26	95.10	Good accuracy
LogisticRegression	$1e{-}5$ (low)	0.71	95.30	No significant improvement

Table 3: Effect of tolerance (tol) on LinearSVC and LogisticRegression

Model	Penalty	Train Time (s)	Test Accuracy (%)	Remarks
LinearSVC	l2	0.36	94.00	Preferred, stable solution
LinearSVC	l1	0.48	92.80	Needs dual=False, slower
LogisticRegression	l2	0.26	95.10	Best result
LogisticRegression	l1	0.42	93.70	Sparse, slightly worse

Table 4: Effect of penalty type (l1 vs l2) on LinearSVC and LogisticRegression