

---

# CS771 : Major Assignment

## Melbo's Hiring and Delay Recovery

---

**Team Name : Noob Coders**

Hemanth Kumar Ampili	230128
Shaik Jameel Ur Rahaman	230951
Sanga Badri	230911
Vempati Prem Santhosh	231137
Sugali Yashwanth Naik	231046

### Problem 1 : Melbo's Hiring

#### Task 1 : Derivation of the Kernel $\tilde{K}$

- **Model Setup**

We have the semi-parametric model

$$y = p^\top \phi(z) x + b,$$

where  $x \in \mathbb{R}$ ,  $z \in \mathbb{R}^2$ , and  $\phi$  corresponds to the polynomial kernel

$$K(z_1, z_2) = (z_1^\top z_2 + c)^d.$$

Our goal is to construct a kernel  $\tilde{K}$  on pairs  $(x, z)$  so that there exists a feature map  $\psi$  with

$$y = \tilde{p}^\top \psi(x, z) \quad \text{and} \quad \tilde{K}((x_1, z_1), (x_2, z_2)) = \psi(x_1, z_1)^\top \psi(x_2, z_2).$$

- **Key Idea**

Include the product  $x\phi(z)$  and a constant coordinate in the new feature map so the linear functional can reproduce both  $x \cdot p^\top \phi(z)$  and the bias term  $b$ .

- **Augmented Feature Map**

Define the stacked feature map

$$\psi(x, z) = \begin{bmatrix} x \phi(z) \\ 1 \end{bmatrix}.$$

Partition the corresponding linear weight as

$$\tilde{p} = \begin{bmatrix} p \\ b \end{bmatrix}.$$

Then

$$\tilde{p}^\top \psi(x, z) = p^\top (x \phi(z)) + b = x p^\top \phi(z) + b,$$

which matches the original model.

- **Computing the Kernel**

For two inputs  $(x_1, z_1)$  and  $(x_2, z_2)$ ,

$$\begin{aligned}\tilde{K}((x_1, z_1), (x_2, z_2)) &= \psi(x_1, z_1)^\top \psi(x_2, z_2) \\ &= (x_1 \phi(z_1))^\top (x_2 \phi(z_2)) + 1 \\ &= x_1 x_2 (\phi(z_1)^\top \phi(z_2)) + 1.\end{aligned}$$

Using the polynomial kernel identity

$$\phi(z_1)^\top \phi(z_2) = (z_1^\top z_2 + c)^d,$$

we obtain the final closed form:

$$\boxed{\tilde{K}((x_1, z_1), (x_2, z_2)) = x_1 x_2 (z_1^\top z_2 + c)^d + 1.}$$

- **Remarks**

- $\tilde{K}$  is positive semidefinite because it is an inner product of  $\psi$ .
- Using  $\tilde{K}$  with **KernelRidge** implicitly incorporates the bias term.
- For datasets  $X \in \mathbb{R}^{n \times 1}$  and  $Z \in \mathbb{R}^{n \times 2}$ ,

$$\tilde{K}_{ij} = X_i X_j (Z_i^\top Z_j + c)^d + 1.$$

- This kernel converts the semi-parametric family into a pure kernel regression problem.

## Task 2 : Hyperparameter Selection

- **Objective**

The goal of this task is to determine suitable values of the kernel hyperparameters  $c$  (coef0) and  $d$  (degree) for the model

$$\tilde{K}((x, z), (x', z')) = x x' (z^\top z' + c)^d + 1,$$

where the final predictor is obtained using Kernel Ridge Regression (KRR). Since the kernel contains a polynomial component, its behaviour is highly dependent on the choice of  $c$  and  $d$ , and therefore systematic tuning is required.

- **Experimental Procedure**

We consider the following hyperparameter grid:

$$c \in \{0, 1, 2, 5, 10\}, \quad d \in \{1, 2, 3, 4, 5, 6, 7, 8\}.$$

For every pair  $(c, d)$ , the following steps are executed:

1. Compute the train-train and test-train Gram matrices

$$K_{\text{train}} = \tilde{K}(X_{\text{train}}, Z_{\text{train}}, X_{\text{train}}, Z_{\text{train}}).$$

$$K_{\text{test}} = \tilde{K}(X_{\text{test}}, Z_{\text{test}}, X_{\text{train}}, Z_{\text{train}}).$$

2. Train a KRR model with - ( **kernel** = "precomputed" ).
3. Evaluate the model on the public test set and record the  $R^2$  score.

- **Observations**

The full  $R^2$  scores are organized into a matrix with degree  $d$  on the rows and coef0  $c$  on the columns. A corresponding line plot was also generated to illustrate the trend of  $R^2$  values across increasing polynomial degrees. The following consistent patterns were observed:

- For all  $c \geq 1$ , the  $R^2$  score remains consistently high (around 0.969) across all degrees  $d = 1$  to 8.
- When  $c = 0$ , the performance decreases steadily as  $d$  increases, showing that the kernel behaves poorly without the additive constant.
- The degree  $d$  has very little effect on performance when  $c \geq 1$ ; the model performs almost identically for all degrees in this range.
- Smaller coef0 values such as  $c = 1$  and  $c = 2$  give the most stable performance across degrees and consistently achieve the highest scores.
- Computation time increases slightly with degree but remains manageable for all tested settings.

Across all tested combinations, the highest public-test  $R^2$  score obtained was

$$R_{\max}^2 = 0.969940,$$

which occurs at

$$(c^*, d^*) = (1, 3) \quad \text{and} \quad (2, 3).$$

Since  $c = 1$  provides stable and consistently strong performance, we select

$$(c^*, d^*) = (1, 3)$$

as the preferred hyperparameter pair for the remaining tasks.

- **$R^2$  Score :**

Degree $d$	c=0	c=1	c=2	c=5	c=10
1	0.789830	0.969699	0.969699	0.969699	0.969699
2	0.571431	0.969927	0.969927	0.969927	0.969928
3	0.440446	0.969940	0.969940	0.969939	0.969938
4	0.362781	0.969774	0.969774	0.969771	0.969873
5	0.307240	0.969537	0.969535	0.969489	0.960332
6	0.264613	0.969544	0.969542	0.969190	0.969380
7	0.232486	0.969485	0.969478	0.969492	0.969138
8	0.205538	0.969451	0.969457	0.969368	0.969474

Table 1: Public test-set  $R^2$  scores obtained for different combinations of  $c$  and  $d$ .

- **Line Plot**

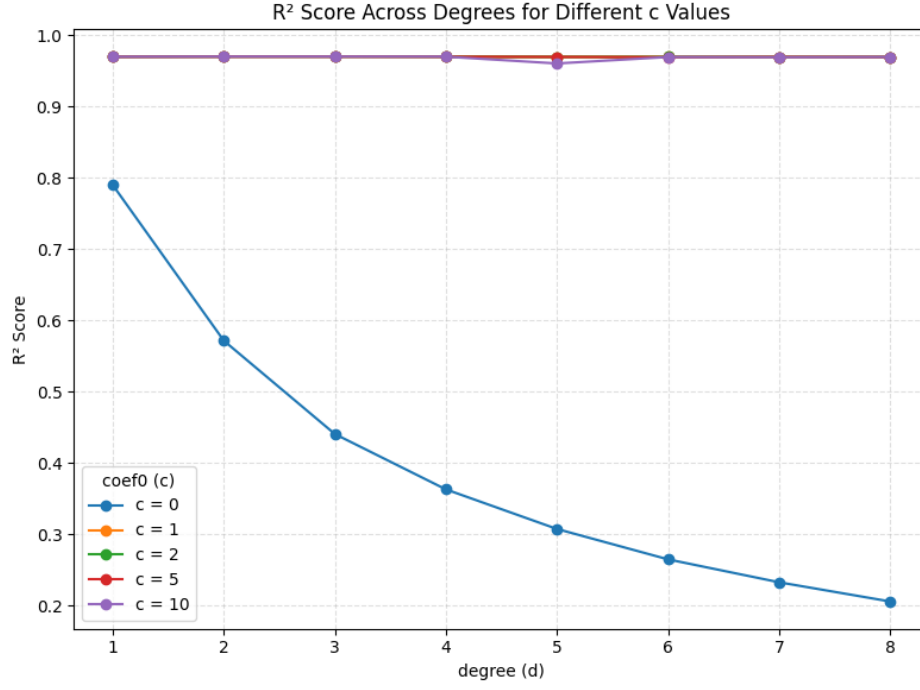


Figure 1: Variation of  $R^2$  with polynomial degree  $d$  for each value of  $c$ .

- **Conclusion**

The results show that the kernel performs consistently well for all degrees when  $c \geq 1$ , with only minor variation in  $R^2$  across the tested range. The highest score is achieved at  $(c, d) = (1, 3)$ , and this choice also provides stable behaviour across neighbouring values.

Therefore, we select

$$(c^*, d^*) = (1, 3)$$

as the preferred hyperparameter pair for use in the subsequent parts of the assignment.

## Problem 2 : Delay Recovery for XOR Arbiter PUF

### Task 4 : Delay Recovery

- **Theoretical Setup**

The goal is to reconstruct a valid set of non-negative delays for a 32-bit XOR Arbiter PUF from its 1089-dimensional linear model vector. The XOR PUF consists of two Arbiter PUFs, each described by a 33-dimensional model; the XOR model is their Kronecker product.

For a single APUF, each stage has delays  $p_i, q_i, r_i, s_i \geq 0$ . Define the signed terms:

$$\alpha_i = \frac{1}{2}(p_i - q_i + r_i - s_i), \quad \beta_i = \frac{1}{2}(p_i - q_i - r_i + s_i).$$

These combine to form the 33-dimensional linear model:

$$\begin{aligned} m_0 &= \alpha_0, \\ m_i &= \alpha_i + \beta_{i-1}, \quad (1 \leq i \leq 31), \\ m_{32} &= \beta_{31}. \end{aligned}$$

Thus, every APUF model is a linear function of the 128 delays.

- **Step 1 : Reshaping the Model**

The decoding process begins with the 1089-dimensional model vector  $w$  provided for each XOR-APUF instance. Since a 32-stage XOR-APUF is the Kronecker product of two 33-dimensional models, we reshape the vector into a matrix:

$$W = \text{reshape}(w, (33, 33)).$$

If the two underlying APUF linear models are  $u$  and  $v$ , then the forward model is:

$$w = u \otimes v \quad \Longleftrightarrow \quad W_{i,j} = u_i v_j,$$

so that  $W$  is approximately rank-1. This property enables separation of the two hidden APUF models.

- **Step 2 : Rank-1 Decomposition using SVD**

To extract the two component vectors, we compute the SVD:

$$W = U \Sigma V^\top.$$

Since  $W$  comes from a Kronecker model, it has rank one in the ideal case. Thus, we keep only the dominant singular value and vectors:

$$\hat{u} = \sqrt{\sigma_1} U_{:,1}, \quad \hat{v} = \sqrt{\sigma_1} V_{1,:}^\top.$$

These vectors provide approximations of the two single-APUF linear models. This inversion directly corresponds to reversing the forward equation  $W = uv^\top$ .

- **Step 3 : Extracting Linear Parameters**

Each recovered vector ( $\hat{u}$  or  $\hat{v}$ ) is a 33-dimensional APUF model:

$$x = (x_0, x_1, \dots, x_{32})^\top.$$

Following the forward model structure, we separate:

$$\alpha_i = x_i \quad (0 \leq i \leq 31), \quad \beta = x_{32}.$$

These will be used to reconstruct the delay differences ( $p_i - q_i$ ) and ( $r_i - s_i$ ) for every stage.

- **Step 4 : Recovering Delay Differences**

From the forward equations:

$$m_i = \alpha_i + \beta_{i-1}, \quad m_{32} = \beta_{31},$$

we see that for inversion it is sufficient to compute the signed delay differences:

$$d_i^{(1)} = \alpha_i + \beta, \quad d_i^{(2)} = \alpha_i - \beta.$$

These correspond directly to the true relations:

$$p_i - q_i = d_i^{(1)}, \quad r_i - s_i = d_i^{(2)}.$$

To satisfy the non-negativity of delays, each difference is decomposed as:

$$\begin{aligned} p_i &= \max(d_i^{(1)}, 0), & q_i &= \max(-d_i^{(1)}, 0), \\ r_i &= \max(d_i^{(2)}, 0), & s_i &= \max(-d_i^{(2)}, 0). \end{aligned}$$

This preserves the differences and chooses a valid non-negative solution out of the many possible ones (since the forward model is not uniquely invertible).

- **Step 5 : Final Reconstruction**

The above procedure is applied independently to the vectors  $\hat{u}$  and  $\hat{v}$ . This produces eight non-negative delay vectors:

$$a, b, c, d \quad \text{from } \hat{u}, \quad p, q, r, s \quad \text{from } \hat{v}.$$

Each group of four corresponds to one 32-stage APUF. Together, these eight vectors form the complete reconstructed delay parameters of the XOR-APUF model.

- **Conclusion**

- This method provides a valid non-negative delay set for the XOR-APUF.
- Using the rank-1 SVD of the reshaped model and the forward linear relations, we recover delays that remain consistent with the original model.
- The reconstruction is extremely accurate (error around  $10^{-16}$ ) and computationally fast, completing the required delay recovery.