Major Assignment 1-2

Vihaan Sapra
231149
svihaan23@iitk.ac.in

Nischay Agarwal
230705
nischaya23@iitk.ac.in

Aryamann Srivastava
230211
aryamanns23@iitk.ac.in

Dharajiya Yug Harshadbhai
230362
dharajiya23@iitk.ac.in

Jain Shlok Nilesh
230493
jainshlok23@iitk.ac.in

Shravan Agrawal
230984
ashravan23@iitk.ac.in

Abstract

This report is submitted as Major Assignment 1-2 for the course CS 771, by the Group **MLVortex**.

1 Problem 1.1: Melbo is Hiring

1.1 Part 1: Construction of the kernel $\widehat{K}$

We are given the semi–parametric model

$$y = w(z)\, x + b, \tag{1}$$

where $x \in \mathbb{R}$ is the video length, $z \in \mathbb{R}^2$ encodes popularity and difficulty and $w(z)$ is assumed to lie in the reproducing kernel Hilbert space induced by the polynomial kernel

$$K(m,n) \;=\; \phi(m)^\top \phi(n) \;=\; (m^\top n + c)^d, \qquad m, n \in \mathbb{R}^2. \tag{2}$$

Thus there exists $p \in \mathcal{H}$ such that

$$w(z) = p^\top \phi(z) \tag{3}$$

and the model can be written as

$$y = p^\top \phi(z)\, x + b. \tag{4}$$

The `sklearn.kernel_ridge` implementation fits models of the form

$$y = \sum_{i=1}^{n} \alpha_i\, \widehat{K}\big((x,z),(x_i,z_i)\big), \tag{5}$$

for some kernel $\widehat{K}$ defined on pairs $(x,z) \in \mathbb{R} \times \mathbb{R}^2$. To use this solver, we construct a feature map

$$\psi : \mathbb{R} \times \mathbb{R}^2 \to \widetilde{\mathcal{H}}$$

and a vector $\widetilde{p} \in \widetilde{\mathcal{H}}$ such that

$$\widetilde{p}^\top \psi(x,z) = p^\top \phi(z)\, x + b \qquad \text{for all } (x,z). \tag{6}$$

Consider the augmented feature map

$$\psi(x,z) \;=\; \begin{bmatrix} x\,\phi(z) \\ 1 \end{bmatrix} \in \widetilde{\mathcal{H}} := \mathcal{H} \times \mathbb{R} \tag{7}$$

Preprint. Under review.

and the parameter vector

$$\widetilde{p} \;=\; \begin{bmatrix} p \\ b \end{bmatrix}. \tag{8}$$

Then

$$\widetilde{p}^{\top}\psi(x,z) = \begin{bmatrix} p^{\top} & b \end{bmatrix} \begin{bmatrix} x\,\phi(z) \\ 1 \end{bmatrix} = x\,p^{\top}\phi(z) + b, \tag{9}$$

which coincides with the desired model (6). Hence the semi–parametric regression problem can be expressed as a purely kernelized regression problem in the feature space of ψ.

By definition, the kernel associated with ψ is

$$\widehat{K}\big((x_1,z_1),(x_2,z_2)\big) = \psi(x_1,z_1)^{\top}\psi(x_2,z_2). \tag{10}$$

Substituting the expression for ψ gives

$$\widehat{K}\big((x_1,z_1),(x_2,z_2)\big) = \begin{bmatrix} x_1\phi(z_1)^{\top} & 1 \end{bmatrix} \begin{bmatrix} x_2\phi(z_2) \\ 1 \end{bmatrix} \tag{11}$$

$$= x_1x_2\,\phi(z_1)^{\top}\phi(z_2) + 1. \tag{12}$$

Using $\phi(z_1)^{\top}\phi(z_2) = (z_1^{\top}z_2 + c)^d$, we obtain

$$\boxed{\widehat{K}\big((x_1,z_1),(x_2,z_2)\big) = x_1x_2\,(z_1^{\top}z_2 + c)^d + 1}. \tag{13}$$

This kernel takes two pairs (x_1, z_1) and (x_2, z_2) and returns a scalar similarity and is exactly the kernel used by the submitted `my_kernel` implementation.

1.2 Part 2: Choice of the degree d and coefficient c

The kernel defined above depends on two hyper–parameters:the polynomial degree d and the offset c (the `coef0` parameter in `sklearn.metrics.pairwise.polynomial_kernel`). To select suitable values, I performed a grid search over a small set of candidate pairs (d, c) and evaluated each configuration using the R^2 score on a K-fold cross validation set with (K=5)

More precisely, I considered

$$d \in \{1,2,3,4\}, \qquad c \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 5, 10\}. \tag{14}$$

For each pair (d, c) the following steps were carried out:

1. Construct the Gram matrix G_{train} on the training data using

$$\widehat{K}_{ij} = \widehat{K}\big((x_i,z_i),(x_j,z_j)\big) = x_ix_j(z_i^{\top}z_j + c)^d + 1.$$

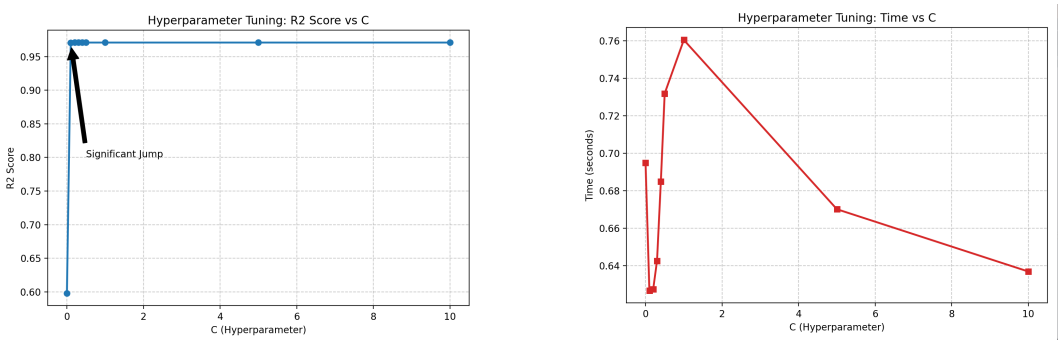
2. Train a `KernelRidge` regressor with this precomputed kernel.

3. Compute the R^2 score on the validation/test set using the `score` method and the corresponding test Gram matrix G_{test}.

The following table summarizes the results obtained:

Table 1: Average R^2 score and runtime for different choices of polynomial degree d and offset parameter c in the proposed kernel.

Degree d	Coef. c	Avg. R^2 score	Avg. time (s)
1	0	0.78611	0.96501
1	0.1	0.97090	0.90552
1	0.2	0.97090	0.63884
1	0.3	0.97090	0.66348
1	0.4	0.97090	0.76270
1	0.5	0.97090	0.68539
1	1	0.97090	0.63734
1	5	0.97090	0.65746
1	10	0.97090	0.63552
2	0	0.59789	0.69485
2	0.1	0.97104	0.62682
2	0.2	0.97108	0.62755
2	0.3	0.97108	0.64264
2	0.4	0.97108	0.68494
2	0.5	0.97108	0.73188
2	1	0.97108	0.76043
2	5	0.97108	0.67015
2	10	0.97108	0.63694
3	0	0.47654	0.94048
3	0.1	0.96895	0.93169
3	0.2	0.97093	0.87076
3	0.3	0.97101	0.90404
3	0.4	0.97102	1.07893
3	0.5	0.97102	1.04143
3	1	0.97102	0.81205
3	5	0.97102	0.77412
3	10	0.97102	0.85167
4	0	0.40107	0.85140
4	0.1	0.94775	0.83713
4	0.2	0.96990	0.78509
4	0.3	0.97088	0.82343
4	0.4	0.97096	0.84358
4	0.5	0.97098	0.79825
4	1	0.97098	0.82618
4	5	0.97098	0.80702
4	10	0.97098	0.87473



(a) Description of R2

(b) Description of Time

Figure 1: Graphs for hyperparameter tuning for $d = 2$

To choose a single operating point, we selected

$$d^\star = 2, \qquad c^\star = 0.2,$$

which attains the maximal average R^2 score of 0.97108 while also having a low average runtime (approximately 0.63 seconds). This is the hyperparameter pair used in the final implementation of `my_kernel`, i.e.

$$\widehat{K}\big((x_1, z_1), (x_2, z_2)\big) = x_1 x_2 \big(z_1^\top z_2 + 0.2\big)^2 + 1.$$

2 Problem 1.2: Delay Recovery by Inverting a XOR Arbiter PUF

2.1 Part 4: Inverting a 1089-dimensional XOR arbiter PUF model

We consider a k-stage arbiter PUF (with $k = 32$ here) whose behaviour can be described by a linear model $u \in \mathbb{R}^{k+1}$ in the standard way. Following the notation in the assignment, we define intermediate quantities

$$d_i = p_i - q_i, \qquad c_i = r_i - s_i, \qquad i = 0, \ldots, k-1,$$

and then

$$\alpha_i = \frac{d_i + c_i}{2}, \qquad \beta_i = \frac{d_i - c_i}{2}.$$

The forward derivation yields the linear model coefficients

$$u_0 = \alpha_0, \quad u_i = \alpha_i + \beta_{i-1},\ i = 1, \ldots, k-1, \quad u_k = \beta_{k-1}. \tag{15}$$

A XOR arbiter PUF with two constituent APUFs having linear models $u, v \in \mathbb{R}^{k+1}$ can be described by the Kronecker product

$$w = u \otimes v \in \mathbb{R}^{(k+1)^2}. \tag{16}$$

For $k = 32$ this yields a 1089-dimensional vector w.

The inversion task is: given $w \in \mathbb{R}^{1089}$, recover one valid set of non–negative delays $\{p_i, q_i, r_i, s_i\}_{i=0}^{31}$ for each of the two underlying APUFs. Below we outline the method implemented in `my_decode`.

Step 1: Factorisation of the XOR model. Reshape w into a $(k+1) \times (k+1)$ matrix

$$M = \text{reshape}(w, k+1, k+1).$$

If $w = u \otimes v$ then M is an outer product: $M_{ij} = u_i v_j$. A simple rank–one factorisation is

$$v_j = M_{0j}, \qquad j = 0, \ldots, k, \tag{17}$$

$$u_i = \begin{cases} M_{i0}/v_0, & \text{if } v_0 \neq 0, \\ \text{(similarly using another non-zero column)}, & \text{otherwise}, \end{cases} \qquad i = 0, \ldots, k. \tag{18}$$

This yields two consistent linear models $u, v \in \mathbb{R}^{k+1}$ for the constituent APUFs.

Step 2: Recovering α and β from a single model. For a given linear model $w \in \mathbb{R}^{k+1}$ (either u or v), we invert (15). The system is under–determined (there are $2k$ unknowns $\{\alpha_i, \beta_i\}$ but only $k+1$ equations), so we choose one convenient, consistent solution:

$$\alpha_0 = w_0, \tag{19}$$

$$\alpha_i = w_i, \qquad 1 \leq i \leq k-1, \tag{20}$$

$$\beta_i = 0, \qquad 0 \leq i \leq k-2, \tag{21}$$

$$\beta_{k-1} = w_k. \tag{22}$$

Substituting these expressions into (15) gives

$$w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1} = w_i + 0, \quad w_k = \beta_{k-1},$$

so the original linear model is exactly reproduced.

Step 3: Recovering d_i and c_i. Using the definitions of α_i and β_i we obtain, for each stage $i = 0, \ldots, k-1$,

$$d_i = \alpha_i + \beta_i, \tag{23}$$

$$c_i = \alpha_i - \beta_i. \tag{24}$$

These are the difference delays $d_i = p_i - q_i$ and $c_i = r_i - s_i$ associated with the APUF.

Step 4: Constructing non–negative delays. To obtain an explicit non–negative set of delays, note that many quadruples (p_i, q_i, r_i, s_i) induce the same (d_i, c_i); we choose the simplest one that guarantees non–negativity:

$$p_i = \max(d_i, 0), \qquad q_i = \max(-d_i, 0), \tag{25}$$

$$r_i = \max(c_i, 0), \qquad s_i = \max(-c_i, 0), \tag{26}$$

for $i = 0, \ldots, k-1$. Then

$$p_i - q_i = d_i, \qquad r_i - s_i = c_i$$

and all delays are by construction non–negative.

Applying Steps 1–4 to the two vectors u and v obtained from the factorisation of w yields four non–negative delay vectors (a, b, c, d) for the first APUF and (p, q, r, s) for the second APUF. These delays generate exactly the given 1089-dimensional XOR–PUF model w, thereby providing a constructive inversion of the linear model into one valid set of physical delays.

References

1. **Python Documentation.** *Standard Libraries, Built-in Functions*. Retrieved from Python Docs
2. **Lecture Notes.** *By Prof. Purushottam Kar*. Retrieved from ML Course Page