Kernel ridge regression

Omar Ghezzi[1] (matr. 984352),
project for the exam of Statistical methods for machine learning

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

Abstract

This study aims to elucidate the fundamental concepts of the Reproducing Kernel Hilbert Space (RKHS) theory, its relationship with machine learning, and a practical demonstration of kernel ridge regression for predictive purposes. Specifically, we applied kernel ridge regression for predicting the popularity of music tracks on Spotify. To alleviate spatial and time complexity issues, we employed the PLU factorization method and a divide-and-conquer approach. We also implemented the 5-fold best CV estimate as a method to compute the expected accuracy of the predictor generated by the kernel ridge regression, supplying tables showcasing estimates corresponding to various hyperparameter choices. Despite the necessity of using lower training sizes due to memory and processing constraints, leading to smaller accuracy (high bias), the expected accuracy demonstrates a notable improvement, especially when employing the divide-and-conquer technique, as compared to the simple ridge regression algorithm.

1 Introduction

In the upcoming pages, we delve into the theoretical framework and practical application of our project, which utilizes kernel ridge regression on the 'Spotify Tracks Dataset' to predict the popularity of the music tracks. Specifically, in section 2, we will introduce the basic concepts of Reproducing Kernel Hilbert Spaces (RKHS) theory and its relation to machine learning. In section 3, we will address the practical challenges and implementation details of the project. Finally, in section 4, we will showcase the results we have achieved through this approach.

The foundations of this work, especially the theoretical parts, are grounded in the lecture materials from the Statistical Methods for Machine Learning course, supplemented by the following references: (Rudin 2012), (Gretton 2022), (Shawe-Taylor & Cristianini 2004), (Song 2020), (Foster 2023) and (Zhang et al. 2015).

2 Theoretical background

2.1 Feature expansion

Consider a learning task $(\mathcal{D}, \ell)$ exemplified by a dataset $S_m = \{(\boldsymbol{x}_t, y_t\}_{t=1}^m$, where $\forall t = 1, ..., m$, $\boldsymbol{x}_t \in \mathcal{X} \equiv \mathbb{R}^d$, $y_t \in \mathcal{Y} \equiv \mathbb{R}$. Consider a learning algorithm A and define the set of the predictors that can be generated by A on a realization S_m of the random sample $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$ as:

$$\mathcal{H}_m = \{h : \mathcal{X} \to \mathcal{Y} \mid \exists S_m, A(S_m) = h\}$$

Corresponding author(s): University of Milan, omar.ghezzi@studenti.unimi.it

Consider the set of linear predictors (homogeneous linear predictors in $\mathbb{R}^d$:

$$\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y} \mid h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^d\} \quad \text{s.t.} \;\; \mathcal{H}_m \subseteq \mathcal{H}$$

linear predictor are (super)parametric because they are always described by a number of parameters bounded by the number d of features.

Learning a more complex non-linear predictor in the original space (which is linear in a expanded space), aiming to achieve consistency (staring from the parametric case), can be done by *feature expansion*: turn data points $\boldsymbol{x} \in \mathbb{R}^d$ into elements of a (possibly infinite-dimensional) reproducing kernel Hilbert space $\mathcal{H}_K$ such that a predictor $h \in \mathcal{H}_K$ can be described by a (possibly infinite) number of parameters.

2.2 Kernel functions

A *kernel function* is a symmetric and positive semi-definite function $K(\cdot, \cdot)$ of two independent variables, which implements the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_K}$ of a completion $\mathcal{H}_K$ of the space $\text{Imm}(\Phi)$ spanned by the components $K(\boldsymbol{x}, \cdot) = \Phi(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathcal{X}$.

$$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

$$\forall \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}, \;\; (\boldsymbol{x}, \boldsymbol{x'}) \mapsto K(\boldsymbol{x}, \boldsymbol{x'}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x'}) \rangle_{H_k}$$

- Symmetry:

$$K(\boldsymbol{x}, \boldsymbol{x'}) = K(\boldsymbol{x'}, \boldsymbol{x})$$

- positive semi-defined [1]:

$$\big\langle \langle f(\cdot), K(\cdot, \cdot) \rangle, f(\cdot) \big\rangle \geq 0 \;\; \forall f(\cdot) : \mathcal{X} \to \mathbb{R}$$

$$\iint f(\boldsymbol{x}) K(\boldsymbol{x}, \boldsymbol{x'}) f(\boldsymbol{x'}) d\boldsymbol{x}, d\boldsymbol{x'} \geq 0 \;\; \forall f : \mathcal{X} \to \mathbb{R}$$

Given a dataset $S_m = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\}$, a kernel $K(\cdot, \cdot)$ gives rise to a positive semi-definite *Gram matrix* encoding the similarity between each pair of data points $\boldsymbol{x}, \boldsymbol{x'} \in S_m$:

$$\forall m \in \mathbb{N}, \forall \boldsymbol{x}_1, ..., \boldsymbol{x}_m \in \mathcal{X}, \;\; \boldsymbol{K} = [K_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{m \times m}$$

- Symmetry:

$$[K_{i,j}]_{m \times m} = [K_{j,i}]_{m \times m}$$

- positive semi-defined:

$$\boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \geq 0 \;\; \forall \boldsymbol{\alpha} \in \mathbb{R}^m$$

[1] Recalling that an m-sampled function, with possibly $m \to \infty$, can be viewed as an infinite dimensional vector, then a symmetric positive semi-definite bi-dimensional function $K(\boldsymbol{x}, \boldsymbol{x'})$ sampled in $m \in \mathbb{N}$ points, can be intended as a square matrix with infinite order. It hols that:

$$\begin{aligned} \big\langle \langle f(\cdot), K(\cdot, \cdot) \rangle, f(\cdot) \big\rangle &= \lim_{||\Delta x|| \to 0} \sum_{i=1}^{m} \sum_{j=1}^{m} f(\boldsymbol{x}_i) K(\boldsymbol{x}_i, \boldsymbol{x}_j) f(\boldsymbol{x}_j) \Delta x \\ &= \iint f(\boldsymbol{x}) K(\boldsymbol{x}, \boldsymbol{x'}) f(\boldsymbol{x'}) d\boldsymbol{x}, d\boldsymbol{x'} \end{aligned}$$

the reasoning behind using m here, instead of the dimension N (or the cardinality n of a span) of $\mathcal{H}_K$, will become clear when we discuss the 'representer theorem' in section 2.5

Spectral theorem The matrix $\boldsymbol{K}$ can be diagonalized, i.e there exists a basis $\{\boldsymbol{q}_i\}_{i=1}^m$ of eigenvectors $\boldsymbol{q}_i$ of $\boldsymbol{K}$ along with eigenvalues $\{\lambda_i\}_{i=1}^m$, s.t. $\boldsymbol{K}\boldsymbol{q_i} = \lambda_i \boldsymbol{q_i}$ and its eigendecomposition has the form (Song 2020):

$$\boldsymbol{K} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^T = \begin{pmatrix} \boldsymbol{q}_1 & ... & \boldsymbol{q}_m \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ . & . & ... & . \\ . & . & ... & . \\ 0 & 0 & ... & \lambda_m \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_1^T \\ . \\ . \\ . \\ \boldsymbol{q}_m^T \end{pmatrix} = \sum_{i=1}^m \lambda_i \boldsymbol{q}_i \boldsymbol{q}_i^T$$

Mercer's theorem Similarly to the matrix case, it is possible to find infinite eigenfunctions $\{\phi_i\}_{i=1}^\infty$ and corresponding eigenvalues $\{\lambda_i\}_{i=1}^\infty$ s.t $\int K(\boldsymbol{x}, \boldsymbol{x'})\phi(\boldsymbol{x})d\boldsymbol{x} = \lambda\phi(\boldsymbol{x'})$. The eigendecomposition has the form (Shawe-Taylor & Cristianini 2004):

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{i=1}^\infty \lambda_i \phi_i(\boldsymbol{x})\phi_i(\boldsymbol{x'})$$

2.3 Feature mapping

Let $\mathbb{R}^{\mathcal{X}}$ be the space of the functions mapping $\mathcal{X}$ to $\mathbb{R}$ (Rudin 2012):

$$\mathbb{R}^{\mathcal{X}} = \{f : \mathcal{X} \to \mathbb{R}\}$$

Define a *feature map* Φ as an entity that turns each data point $\boldsymbol{x} \in \mathcal{X}$ into a function in $\mathbb{R}^{\mathcal{X}}$:

$$\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$$

$$\boldsymbol{x} \mapsto \Phi(\boldsymbol{x}) = K(\cdot, \boldsymbol{x}) \in \mathbb{R}^{\mathcal{X}}$$

According to the definition, $\Phi(\boldsymbol{x})$ is a function, i.e. an element in a *function space* $\mathbb{R}^{\mathcal{X}}$. We can think of $\mathbb{R}^{\mathcal{X}}$ as a vector space in which each element $f(\cdot)$ is an infinite dimensional vector whose components are infinitesimally close (i.e. $f(\cdot)$ is a continuous-time signal, as typically encountered in signal processing (Song 2020)).[2]

Image space/feature space The image $\text{Imm}(\Phi)$ (*feature space*) of the entity Φ is a subspace[3] of the function space $\mathbb{R}^{\mathcal{X}}$ defined as:

$$\text{Imm}(\Phi) = \left\{ f(\cdot) \in \mathbb{R}^{\mathcal{X}} : \exists \boldsymbol{x} \in \mathcal{X}, f(\boldsymbol{x}) = \Phi(\boldsymbol{x}) \right\}$$

equally, the image of Φ can be defined also as the subspace of $\mathbb{R}^{\mathcal{X}}$ spanned (generated) by the set of functions $\{\Phi(\boldsymbol{x}_i) = K(\cdot, \boldsymbol{x}_i) : \boldsymbol{x}_i \in \mathcal{X}, i = 1, 2, ..., n, n \in \mathbb{N}\}$, i.e expressing each element in $\text{Imm}(\Phi)$ as

[2] Therefore, the inner product defined between vectors in an inner product space can be extended to a *function space* as the integral between functions, exploiting the Riemann sum $\sum_{i=1}^N f(\boldsymbol{x}_i)g(\boldsymbol{x}_i)\Delta\boldsymbol{x}$, with $\Delta\boldsymbol{x} = \boldsymbol{x}_i - \boldsymbol{x}_{i-1}$:

$$\begin{aligned} \langle f(\cdot), g(\cdot) \rangle &= \lim_{||\Delta x|| \to 0} \sum_{i=1}^N f(\boldsymbol{x}_i)g(\boldsymbol{x}_i)\Delta x \\ &= \int_{\mathcal{X}} f(\boldsymbol{x})g(\boldsymbol{x})d\boldsymbol{x} \end{aligned}$$

[3] Note that $\text{Imm}(\Phi)$ is a subspace of $\mathbb{R}^{\mathcal{X}}$. While $\mathbb{R}^{\mathcal{X}}$ has infinite dimension, $\text{Imm}(\Phi)$ may have either finite or infinite dimension $\text{Dim}(\text{Imm}(\Phi)) = N$, with $N \in \mathbb{N}$.

a linear combination of elements of a specific set of generators for $\mathrm{Imm}(\Phi)$, using suitable coefficients $\boldsymbol{\alpha}$ with $\alpha_i \in \mathbb{R}$, and an arbitrary choice of $\boldsymbol{x}_i \in \mathcal{X}$ for all $i = 1, 2, ..., n \in \mathbb{N}$:[4]

$$\begin{aligned}\mathrm{Imm}(\Phi) &= \mathrm{Span}\big(\{\Phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X})\big\} \\ &= \Big\{ f(\cdot) = \sum_{i=1}^{n} \alpha_i \Phi(\boldsymbol{x}_i) : n \in \mathbb{N}, \boldsymbol{x}_i \in \mathcal{X}, \alpha_i = \mathbb{R} \Big\} \\ &= \Big\{ f(\cdot) = \sum_{i=1}^{n} \alpha_i K(\cdot, \boldsymbol{x}_i) : n \in \mathbb{N}, \boldsymbol{x}_i \in \mathcal{X}, \alpha_i = \mathbb{R} \Big\}\end{aligned}$$

Inner product in Imm(Φ) : Given $f(\cdot), g(\cdot) \in \mathrm{Imm}(\Phi)$, namely $f(\cdot) = \sum_{i=1}^{n} \alpha_i \Phi(\boldsymbol{x}_i)$ and $f(\cdot) = \sum_{j=1}^{n'} \beta_j \Phi(\boldsymbol{x}_j)$, define $\langle \cdot, \cdot \rangle_{\mathrm{Imm}(\Phi)}$ as:

$$\langle f(\cdot), g(\cdot) \rangle_{\mathrm{Imm}(\Phi)} = \sum_{i=i}^{n} \sum_{i=j}^{n'} \alpha_i \beta_j K(\boldsymbol{x_i}, \boldsymbol{x}_j)$$

it is symmetric, bilinear, and strictly positive definite. Therefore, the feature space $\mathrm{Imm}(\Phi)$ is an inner product space (i.e. the operator $\langle \cdot, \cdot \rangle_{\mathrm{Imm}(\Phi)}$ is well defined).

2.4 Reproducing kernel Hilbert spaces

RKHS Given a (compact) data space $\mathcal{X}$, a reproducing kernel Hilbert space $\mathcal{H}_K$ is an Hilbert space defined as the completion[5] of the inner product space $\mathrm{Imm}(\Phi)$, spanned by $\{K(\boldsymbol{x}_i, \cdot) : \boldsymbol{x}_i \in \mathcal{X}\}$, in $\mathbb{R}^{\mathcal{X}}$:

$$\mathcal{H}_K = \mathrm{Imm}(\Phi) \bigcup \Big\{ h(\cdot) : h(\cdot) = \lim_{n \to \infty} \{f_n(\cdot)\}, \;\; f_1(\cdot), f_2(\cdot), ... \in \mathrm{Imm}(\Phi) \Big\}$$

Namely, an Hilbert space $\mathcal{H}_K$ is a RKHS if there exists $K : \mathcal{X} \to \mathbb{R}$ s.t.:

1. K satisfies the reproducing property, i.e. $\langle f(\cdot), \Phi(\boldsymbol{x}) \rangle_{\mathcal{H}_K} = f(\boldsymbol{x})$.
2. K spans $\mathcal{H}_K$, i.e. $\mathcal{H}_K = \mathrm{Span}\{K(\cdot, \boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X}\}$

Reproducing property Assuming $\{\lambda_i \phi_i\}_{i=1}^{\infty}$ to be a basis of the completion[6] $\mathcal{H}_K$ of $\mathrm{Imm}(\Phi)$ in $\mathbb{R}^{\mathcal{X}}$, define the function $\Phi(\boldsymbol{x}) = K(\boldsymbol{x}, \cdot) \in \mathcal{H}_K$ as:

$$\Phi(\boldsymbol{x}) = K(\boldsymbol{x}, \cdot) = (\sqrt{\lambda_1}\phi_1(\boldsymbol{x}), \sqrt{\lambda_2}\phi_2(\boldsymbol{x})), ...)$$

By definition of $\langle \cdot, \cdot \rangle_{\mathcal{H}_K}$, considering $f(\cdot) = K(\boldsymbol{x}, \cdot)$ and $g(\cdot) = K(\boldsymbol{x}', \cdot)$:

$$\langle K(\boldsymbol{x}, \cdot), K(\boldsymbol{x}', \cdot) \rangle_{\mathcal{H}_K} = \sum_{i=1}^{\infty} \sqrt{\lambda_i}\phi_i(\boldsymbol{x}) \sqrt{\lambda_i}\phi_i(\boldsymbol{x}') \tag{1}$$

$$= \sum_{i=1}^{\infty} \lambda_i \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{x}') \tag{2}$$

---

[4] The cardinality $n \in \mathbb{N}$ of a system spanning a space is not necessarily equal to the dimension of the space, nor is the system required to be a basis of the space. However, it must satisfy $N \leq n$ (equality holds when the system of generators consists of linearly independent elements, indicating that it forms a basis for the space).

[5] Since requiring the $\mathrm{Span}\{K(\boldsymbol{x}_i, \cdot), i = 1, ..., N\}$ to be $\mathcal{H}_K$, other than $\mathrm{Imm}(\Phi)$, is too demanding, the mathematical process of completion is applied to achieve the same aim ($\mathcal{H}_K$ completes $\mathrm{Imm}(\Phi)$).

[6] $\mathrm{Imm}(\Phi)$ is enriched to include all limits $h \in \mathbb{R}^{\mathcal{X}}$ of the Cauchy sequences $\{f_n\} \in \mathrm{Imm}(\Phi)$ that may not be originally included in the image of Φ.

recalling that, by Mercer's theorem, the spectral representation of the kernel has the form:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{x}')$$

We have found that the kernel function is a way to compute the inner product between elements of the span of $\mathcal{H}_K$:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle K(\boldsymbol{x}, \cdot), K(\boldsymbol{x}', \cdot) \rangle_{\mathcal{H}_K} = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle_{\mathcal{H}_K}$$

We can observe that the previous result is a particular instance of the *reproducing property*. This property directly arises from the definition of $\langle \cdot, \cdot \rangle_{\mathcal{H}_K}$.

$$\langle f(\cdot), \Phi(\boldsymbol{x}) \rangle_{\mathcal{H}_K} = f(\boldsymbol{x})$$

$$\langle f(\cdot), \Phi(\boldsymbol{x}) \rangle_{\mathcal{H}_K} = \langle \sum_{i=i}^{n} \alpha_i \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}) \rangle_{\mathcal{H}_K} = \sum_{i=i}^{n} \alpha_i \langle K(\boldsymbol{x}_i,), \Phi(\boldsymbol{x}) \rangle_{\mathcal{H}_K} = \quad (\text{since} \langle \cdot, \cdot \rangle_{\mathcal{H}_K} \text{ is bilinear})$$

$$= \sum_{i=i}^{n} \alpha_i \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}) \rangle_{\mathcal{H}_K} = \sum_{i=i}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) = f(\boldsymbol{x}) \quad (\text{due to Riesz representation theorem})$$

with $n \in \mathbb{N}$ be the cardinality of the arbitrary span of $\mathcal{H}_K$.

Rizes representation theorem In a RKHS $\mathcal{H}_K$, evaluation functionals can always be represented as inner products (Gretton 2022), for each $\boldsymbol{x} \in \mathcal{X}$. In other words, consider expressing $f(\cdot) \mapsto f(\boldsymbol{x})$ as the Dirac delta evaluation functional $\delta_{\boldsymbol{x}}$, namely the mapping that allows each $f(\cdot) \in \mathcal{H}_K$ to be sampled at specific points $\boldsymbol{x}$:

$$\delta_{\boldsymbol{x}} : \mathcal{H}_K \to \mathbb{R}$$

$$\forall f(\cdot) \in \mathcal{H}_K, \;\; \delta_{\boldsymbol{x}} \mapsto f(\boldsymbol{x}) = \delta_{\boldsymbol{x}} f(\cdot)$$

then the Riesz representation theorem can be formalized as:

$$\exists f_{\delta_{\boldsymbol{x}}}(\cdot) \in \mathcal{H}_K \;\; : \;\; \delta_{\boldsymbol{x}} f(\cdot) = \langle f(\cdot), f_{\delta_{\boldsymbol{x}}}(\cdot) \rangle_{\mathcal{H}_K}, \;\; \forall f(\cdot) \in \mathcal{H}_K$$

clearly, setting $f_{\delta_{\boldsymbol{x}}}(\cdot) = K(\cdot, \boldsymbol{x})$ and writing $f(\cdot)$ as the linear combination $f(\cdot) = \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}_i, \cdot)$, we obtain:

$$f(\boldsymbol{x}) = \delta_{\boldsymbol{x}} f(\cdot) = \langle f(\cdot), f_{\delta_{\boldsymbol{x}}}(\cdot) \rangle_{\mathcal{H}_K} = \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$$

which was exactly the last step in expliciting the reproducing property.

2.5 Representer theorem and kernel ridge regression

As we mentioned above, learning a more complex non-linear predictor in the original space (which is linear in a expanded space) can be achieved through feature expansion. Up to now, we saw that to build a RKHS $\mathcal{H}_K$ from a system of generators or to express each function of $\mathcal{H}_K$ evaluated in a single point $\boldsymbol{x} \in \mathcal{X}$ one needs at least $n \in \mathbb{N}$ components, which might be infinite. However, we are interested not in all elements $\mathcal{H}_K$ but in the one that a learning algorithm will choose following a certain criterion, where this choice is formalized as an optimization problem. The *representer theorem* states that even if we're trying to solve an optimization problem in an infinite dimensional space $\mathcal{H}_K$ containing linear combinations of functions $K(\boldsymbol{x}_i, \cdot)$ centered in $n \in \mathbb{N}$ arbitrary points $\boldsymbol{x}_i$, if we take the training points $\boldsymbol{x}_t$ (s.t. $(\boldsymbol{x}_t, y_t) \in S_m$) as those arbitrary points, then the solution lies in the span of the m functions $K(\boldsymbol{x}_t, \cdot)$ centered in the $\boldsymbol{x}_t$, $t = 1, ..., m$.

Representer theorem Given a learning problem $(\mathcal{D}, \ell)$, represented by a realization $S_m = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^m$, where $\boldsymbol{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$, let K be a kernel function and $\mathcal{H}_K$ be the corresponding RKHS, then the solution of the regularized ERM run on $\mathcal{H}_K$ as the hypothesis class, namely:

$$f^*(\cdot) = \underset{f(\cdot)\in\mathcal{H}_K}{\arg\min} \sum_{t=1}^{m} \ell(f(\boldsymbol{x}_i), y_i) + \lambda ||f(\cdot)||^2_{\mathcal{H}_K}$$

has the form:

$$f^*(\cdot) = \sum_{t=1}^{m} \alpha_t K(\boldsymbol{x}_t, \cdot)$$

We can prove it

Where **??** holds for the reproducing property of $\mathcal{H}_K$ and since $\langle f_\perp(\cdot), K(\boldsymbol{x}_t, \cdot)\rangle_{\mathcal{H}_K} = 0$, $\forall t = 1, .., m$, by definition of orthogonality. Therefore the regularized ERM on $\mathcal{H}_K$ becomes the problem of finding $\boldsymbol{\alpha}^*_{S_m}$ s.t:

$$\boldsymbol{\alpha}^*_{S_m} = \underset{\alpha\in\mathbb{R}^m}{\arg\min} \sum_{t=1}^{m} \ell\Big(\sum_{t'=1}^{m} \alpha_{t'} K(\boldsymbol{x}_t, \boldsymbol{x}_{t'}), y_i\Big) + \lambda || \sum_{t'=1}^{m} \alpha_{t'} K(\cdot, \boldsymbol{x}_{t'})||^2_{\mathcal{H}_K}$$

Kernel ridge regression Exploiting the feature mapping technique, the representer theorem allows the representation of the linear predictor $\boldsymbol{w}^*_{S_m}$ as a function in a reproducing kernel hilbert space $\mathcal{H}_K$ described by the parameters $\boldsymbol{\alpha}^* \in \mathbb{R}^m$. Therefore, kernel ridge regression is a non-parametric algorithm: as m increases unbounded, also the number of parameters $\boldsymbol{\alpha}^*$ can grow sub-linearly, providing not only a way to decrease the estimation error (the regularization term) but also a way to potentially reduce the approximation error (as the complexity of the new hypothesis class $\mathcal{H}_K$ can fit the complexity of the phenomenum to be modelled).

Parametric ridge regression has the following closed form solution:

$$\boldsymbol{w}_{S_m} = (\lambda \mathbf{I}_{d\times d} + S^T S)^{-1} S^T \boldsymbol{y}$$

where:

- $S^T = [\boldsymbol{x}_1, ..., \boldsymbol{x}_m]_{d\times m}$ is the transposed version of the **design matrix** S, composed by the m datapoints in the training set S_m.
- $S^T S = [\boldsymbol{x}_i \boldsymbol{x}_j]_{d\times d}$ is a $d \times d$ symmetric matrix.
- $y = [y_1, ..., y_m]^T$ is the vector composed by the m labels in S_m.

exploiting the identity:

$$(\lambda \mathbf{I}_{d\times d} + S^T S)^{-1} S^T = S^T (\lambda \mathbf{I}_{m\times m} + S S^T)^{-1}$$

We can write the kernel ridge regression produced prediction:

$$f^*(\boldsymbol{x}) = \langle \sum_{t=1}^{m} \alpha^*_t K(\boldsymbol{x}_t, \cdot), K(\cdot, \boldsymbol{x})\rangle_{\mathcal{H}_K} = \boldsymbol{y}^T (\lambda \mathbf{I}_{m\times m} + \boldsymbol{K})^{-1} \boldsymbol{K}_{\boldsymbol{x}'} \tag{3}$$

from which we can identify:

$$\boldsymbol{\alpha}^*_{S_m} = \boldsymbol{y}^T (\lambda \mathbf{I} + \boldsymbol{K})^{-1} \quad , \qquad \text{so that:} \qquad f^*(\boldsymbol{x}) = \boldsymbol{\alpha}^{*T}_{S_m} \boldsymbol{K}_{\boldsymbol{x}'}$$

where:

- $SS^T \rightarrow \boldsymbol{K} = [K_{t',t} = K(\boldsymbol{x}_{t'}, \boldsymbol{x}_t)]_{m\times m}$ is the $m \times m$ Gram matrix described in eq.(**??**)
- $S\boldsymbol{x}' \rightarrow \boldsymbol{K}_{\boldsymbol{x}'} = [K_t = K(\boldsymbol{x}_t, \boldsymbol{x})]_{m\times 1}$ is the inner product between each mapped training point and the mapped version of the test instance $\boldsymbol{x}'$

2.6 Gaussian kernel

The **Gaussian kernel** is a kernel function that has the form:

$$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

$$\forall \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}, \;\; (\boldsymbol{x}, \boldsymbol{x'}) \mapsto K(\boldsymbol{x}, \boldsymbol{x'}) = e^{-\frac{1}{2\gamma}||\boldsymbol{x}-\boldsymbol{x'}||^2}$$

The Gaussian kernel is a linear combination of infinitely many polynomial kernels of increasing degree:

$$\begin{aligned}
\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}, \;\; \boldsymbol{x}, \boldsymbol{x}' \mapsto K(\boldsymbol{x}, \boldsymbol{x}') &= e^{-\frac{1}{2\gamma}||\boldsymbol{x}-\boldsymbol{x'}||^2} \\
&= e^{-\frac{||\boldsymbol{x}||^2}{2\gamma}} e^{-\frac{||\boldsymbol{x}'||^2}{2\gamma}} e^{-\frac{1}{\gamma}(\boldsymbol{x}^T\boldsymbol{x}')} \\
&= e^{-\frac{||\boldsymbol{x}||^2}{2\gamma}} e^{-\frac{||\boldsymbol{x}'||^2}{2\gamma}} \sum_{n=0}^{\infty} \frac{1}{n!} \frac{(\boldsymbol{x}^T\boldsymbol{x}')^n}{\gamma^n} \qquad \text{(Taylor expansion.)}
\end{aligned}$$

Recalling that the polynomial kernel has the form $K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T\boldsymbol{x}')^n = \sum_{k=0}^{n} \binom{n}{k} (\boldsymbol{x}^T\boldsymbol{x}')^k$ we can grasp the similarity among the terms that involve $(\boldsymbol{x}^T\boldsymbol{x}')^{(\cdot)}$.

If kernel ridge regression employs the Gaussian kernel, then it becomes a consistent learning algorithm This is due to the fact that the Gaussian kernel is **universal**, i.e. if $K(\cdot, \cdot)$ is the Gaussian kernel, for each $\gamma > 0$ there must exists a function $f^*(\cdot) \in \mathcal{H}_K$ s.t. approximates any continuous function $g : \mathbb{R}^d \to \mathbb{R}$, up to an arbitrary error $\epsilon, \;\; \forall \epsilon > 0$.

3 Methods

3.1 Computational issues

In its standard closed-form implementation, the kernel ridge regression has two main implementation problems:

- the algorithm has to compute the inverted matrix $(\lambda \mathbf{I} + \boldsymbol{K})^{-1}$ which requires $\mathcal{O}(m^3)$ time and $\mathcal{O}(m^2)$ space.
- even if we approximate the kernel matrix or the way we compute its inverse, it is practically impossible to store a $m \times m$ kernel matrix (or a $m' \times m'$ approxiamted one) in a single system with 8GB or 16GB RAM, when m is very big (or m' is still too big).

The way to partially overcome the first issue is to use the **PLU Factorization**. Unfortunately, due to hardware limitations, we can't really overcome the second issue. We can only deal with a sub-sampled $S_{r'}$ version of the original dataset S_r with sample size $r' = 4000$. However, since $r = 83479$ original samples are available, we can still improve the performance of the kernel ridge regression run on r'-sized dataset by using a *divide and conquer* heuristic.

PLU factorization The PLU (permutation-Lower-Upper) decomposition is a way of factorizing a square matrix into the product of a permutation matrix P, a lower triangular matrix L, and an upper triangular matrix U by means of Gaussian-elimination procedures (Foster 2023). Generalizing the LU decomposition, PLU includes a permutation matrix to deal with the situation in which the pivot element, in the Gaussian elimination process, is zero or very close to zero. When this happens, the matrix P simply rearranges the rows of the matrix to bring a non-zero element to the diagonal position:

$$PA = LU$$

where:

- P: permutation matrix, i.e a square binary matrix where each row and each column contains exactly one entry of 1, and all other entries are set to 0.

-

To factorize $(\lambda \mathbf{I} + \boldsymbol{K})$ we use the scipy implementation of the PLU decomposition:

```
P, L, U = sp.linalg.lu(self.lamb*ID + K)
```

We then exploits the scipy method to solve a linear system of equations instead of computing the inverse though 'np.linalg.inv':

```
KID_inv = sp.linalg.solve(U, sp.linalg.solve(L, P))
```

finally we simply compute the parameters $\boldsymbol{\alpha}^*_{S_m} = \boldsymbol{y}^T(\lambda \mathbf{I} + \boldsymbol{K})^{-1}$ applying the the definition:

```
self.alphas = np.dot(y.T, KID_inv).reshape(-1)
```

Divide and conquer A divide and conquer KRR learning algorithm $A = \text{DCKRR}$ implies dividing the training set S_m into $M = \frac{m}{m'}$ equal sized subsets $S_{m'}$, consequently obtaining M training parts $S^{(-i,j)}_{m'}$ for each i-th fold (with $j = 1, ..., M$)[7]. Due to hardware limitations, we are forced to choose only $M = 20$ training parts of sample size $m' \approx 3200$ each. For each iteration of the best CV estimate procedure consists in the following steps (Zhang et al. 2015):

1. Train M predictors $f^{(-i,j)}_\theta(\cdot) = \text{KRR}_\theta(S^{(-i,j)}_{m'})$, each one parametrized by the curent $\theta \in \Theta_0$.
2. Compute the average predictor $\bar{f}^{(-i)}_\theta(\cdot) = \frac{1}{M}\sum_{j=1}^{M} f^{(-i,j)}_\theta(\cdot)$ and test it on the i-th fold[8]

This technique might remind of an ensemble method. However an ensemble methods trains V predictors onto the same full training set S_m, while in the divide and conquer approach we train M predictors on M different down-sampled versions of the original one. In this way we can employ all the original r datapoints, even if we only use m' of them to train predictors.

3.2 Kernel ridge regression implementation

The 'KernelRidgeRegression' is characterized by six attributes:

- **kernel**: type of the chosen kernel (default: 'Gaussian').
- **lambda**: regularization factor λ.
- **gamma**: bandwidth/scale parameter of the Gaussian kernel γ.
- **X**: attribute storing the design matrix S used to compute the kernel matrix $\boldsymbol{K}$ and, partially, the vector $\boldsymbol{K}_{\boldsymbol{x}'}$.
- **alphas**: attribute that will store the parameters $\boldsymbol{\alpha}^*_{S_m}$ of the optimal predictor $f^*(\cdot) \in \mathcal{H}_K$.

The 'RidgeRegression' class includes four principal methods: **fit**, **kernel_matrix**, **predict** and **score**.

Fit: The 'fit' method implements the training phase of the model. It receives the design matrix S (initializing the **X** field) and target vector $\boldsymbol{y}$ from the training set S_m (converting them from pandas dataframes to numpy arrays). It then computes the kernel matrix $\boldsymbol{K}$ calling:

[7] As we employ the 5-fold best CV (a less computationally demanding but statistically less rigorous variant of 5-fold nested cross validation) to estimate the expected risk $\mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\min_{\theta \in \Theta_0} \ell_{\mathcal{D}}\big(\text{DCKRR}_\theta(S_m)\big)\big]$ of $A\big((\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)\big)$, taken with respect to the random draw of $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$

[8] In practice we take each $f^{(-i,j)}_\theta(\cdot)$, we compute the M predictions $f^{(-i,j)}_{(v)\theta}(\boldsymbol{x})$, fixing each datapoint $\boldsymbol{x}$ in the i-th fold $S^{(i)}_n$. We then average the M predictions $\bar{f}^{(-i)}_\theta(\boldsymbol{x}) = \frac{1}{V}\sum_{j=1}^{M} f^{(-i,j)}_{(v)\theta}(\boldsymbol{x})$, so that we can compute the error $\ell_{S^{(i)}_n}\big(\bar{h}^{(-i)}_\theta\big) = \frac{1}{n}\sum_{(\boldsymbol{x},y)\in S^{(i)}_n} \ell\big(y, \bar{h}^{(-i)}_\theta(\boldsymbol{x})\big)$

```
        K = self.kernel_matrix()
```

it then calculates the parameters $\boldsymbol{\alpha}^*_{S_m}$ of the predictor $f^*(\cdot) \in \mathcal{H}_K$ exploiting the PLU factorization.

Kernel_matrix: The 'Kernel_matrix' function computes both $\boldsymbol{K}_{\boldsymbol{x}'}$, if Xp contains the dataframe S' of datapoints belonging to the test set, and $\boldsymbol{K}$, if it receives an empty matrix 'Xp' (in the latter case, it initializes 'Xp' to the class attribute $\mathbf{X}$ containing the design matrix S). Then, if the class attribute 'kernel' is set to 'Gaussian', the method returns:

```
        return np.exp(-1/(2*self.gamma) * np.sum((X[:, np.newaxis] - Xp) ** 2, ...
            axis=-1))
```

Predict: The 'Predict' method implements the prediction phase $\boldsymbol{\alpha}^{*T}_{S_m}\boldsymbol{K}_{\boldsymbol{x}'}$, given new unseen test instances $\boldsymbol{x}'$. It receives the dataframe S' of datapoints belonging to the test set S'_n (converting it into a numpy array). Then, it computes the vector $\boldsymbol{K}_{\boldsymbol{x}'}$ and returns the predicted labels:

```
        Kx = self.kernel_matrix(Xp)
        return np.dot(self.w, Kx)
```

Score: 'scores' assesses the performance of $f^*(\cdot)$ on the test set S_n. The method 'scores' takes as input the test set S'_n, composed by S' and $\boldsymbol{y}'$, it also needs a 'method' specification, either 'mean_squared_error' or 'r2_score' from 'sklearn.metrics', to tell which evaluation metric will be used. It first predicts the labels calling the **predict** method, which generates the vector $\hat{\boldsymbol{y}}'$. Finally, depending on the specified evaluation metric, it calculates the test error or the accuracy of $f^*(\cdot)$[9].

3.3 Divide and conquer kernel ridge regression implementation

The 'DCKRR' class implements the divide and conquer strategy, which allows to store a smaller kernel matrix in a single machine, while still employing all the original data in S_r. The class has the same specification described for the 'KernelRidgeRegression' class, with the only difference regarding the additional hyperparameter M (number of subsets into which divide the whole original training set S_m) and the parameters $\alpha^{*(j)}_{S_{m'}}$ of the M trained models, which are organized into a list of M arrays ('alphas'). The number of subsets M should be an hyperparameter to

Fit: The 'fit' method implements the training phase of the DCKRR model. It receives the design matrix S (initializing the $\mathbf{X}$ field) and target vector $\boldsymbol{y}$ from the training set S_m (converting them from pandas dataframes to numpy arrays). It splits both S and the corresponding $\boldsymbol{y}$ into M subsets and it trains M different instances of the 'KernelRidgeRegression' class (using its 'fit' method). It creates a list for the M parameters $\alpha^{*(j)}_{S_{m'}}$ describing each of the computed predictors.

```
    X_subs = np.array_split(X, self.M)
    y_subs = np.array_split(y, self.M)
```

[9] Indeed, both 'predict' and 'score' can also be used to calculate the predicted labels on the training set and the training error or accuracy $R^2_{S_m}(\boldsymbol{w}S_m)$ (in the previous explanation, simply replace S' with S, $\boldsymbol{y}'$ with $\boldsymbol{y}$, $\hat{\boldsymbol{y}}'$ with $\hat{\boldsymbol{y}}$.

Predict: Calculates the nM predictions applying each j-th fitted model ($j = 1, ..., M$) to each given datapoint $\boldsymbol{x}$ of the test set S_n. This method uses the 'kernel_matrix' method of the 'KernelRidgeRegression' class to compute the predictions:

```
y_hat[j,t] = np.dot(self.alphas[j], model.kernel_matrix(x))
```

Then it averages the predictions obtaining only n values.

Score: the method 'score' assesses the performance of the average predictor $\bar{f}^*(\cdot)$ on the test set S_n. It takes as input the test set S'_n, composed by S' and $\boldsymbol{y}'$ and the 'method' specification ('mean_squared_error' or 'r2_score'). It first predicts the average labels calling the **predict** method of the DCKRR class. Finally, depending on the specified evaluation metric, it calculates the test error or the accuracy of $\bar{f}^*(\cdot)$ (as usual, both 'predict' and 'score' can also be used to calculate the predicted labels on the training set and the training error or accuracy).

4 Results

4.1 Continuous features

In this part, we'll critically analyze the theoretical soundness of the results presented in TABLE 1 and FIG.(1). These results pertain to the application of kernel ridge regression on the dataset $S_r^{(c=1)}$, originally consisting of 83479 examples, but here limited to $r = 4000$ ($m = 3200$ for training).

Given that we've implemented the Gaussian kernel in our model, we understand that for any novel datapoint $\boldsymbol{x} \in \mathcal{X}$, the way kernel ridge regression predicts $f^*(\boldsymbol{x})$ will exhibit similarities to how non-parametric methods (like k-NN) behave as their hyperparameters (the number of nearest neighbors k) change:

- If γ is small ($\gamma \to 0$), then the prediction $f^*(\boldsymbol{x})$ is mainly influenced by the training data points most similar to $\boldsymbol{x}$. As a result, data points in the training set are likely to be predicted precisely with their target (high training accuracy). However, test data points will be predicted based on the targets of only a select few training points, which are most similar to them (small test accuracy). If the size m of the training set is not large enough, this situation typically results in overfitting.
- If γ is high ($\gamma \to \infty$), the prediction $f^*(\boldsymbol{x})$ is influenced by a large number of training points, potentially up to the full size m of the training set (low test accuracy). This means that also the predicted target for each training point will be heavily influenced by the targets of all other points in the training set, resulting in lower training accuracy as well. Essentially, this is a classic case of underfitting.

In our scenario, an inadequate number of examples in the training set (too small m), leads to unavoidable overfitting when γ values are low. Even with high regularization factors, the expected risk cannot converge to the expected training error, across varying random samples. Below we present the interim results obtained from the execution of the 5-fold best CV estimate, considering the following hyperparameters space 'hyper_space':

```
alphas = np.array([0.001, 0.1, 0.5, 1, 5, 10, 50, 100, 1000])
gammas = np.array([0.00001, 0.001, 0.01, 0.1, 1, 2, 4, 5, 10, 15, 20, 50, 100, 500, ...
    4000])
hyper_space = np.array(list(product(alphas, gammas)))
```

Table 1 presents, for each γ_j, the regularization factor $\alpha^{(j)}$ that results in the lowest 'average test error' $\ell_{S_m}^{NCV}(A_{\gamma_j,\theta}(S_m))$ and, consequently, in the highest 'average test accuracy', which is also provided in the table along with the corresponding average training accuracy. The last row of the table is dedicated to the optimal hyperparameters and the estimated accuracy of the optimal predictor. The validation

Gamma	Alpha	Max. training acc. estimate	Max. test acc. estimate
0.00001	1000.0	0.999996	-0.002625
0.00100	1000.0	0.999117	-0.002629
0.01000	1000.0	0.998590	-0.002629
0.10000	1000.0	0.997688	-0.002626
1.00000	10.0	0.774916	0.032003
2.00000	10.0	0.545614	0.044065
4.00000	5.0	0.340123	0.050967
5.00000	5.0	0.290289	0.052553
10.00000	5.0	0.182262	0.054414
15.00000	1.0	0.144970	0.054841
20.00000	1.0	0.126388	0.055644
50.00000	0.5	0.090542	0.054608
100.00000	0.1	0.076321	0.054833
500.00000	0.1	0.045663	0.036767
4000.00000	0.1	0.033401	0.027326
20.0	1.0	0.089439	0.055644

Table 1: Continuous features ($c = 1$). 5-fold best CV estimate, with alpha being chosen within $[\alpha_0, ..., \alpha_8] = [0.00001, 0.001, 0.01, 0.1, 1, 2, 4, 5, 10, 15, 20, 50, 100, 500, 4000]$.

curve for $\alpha = 1000$ and $\gamma = 0.00001$ is depicted in Fig.(1).

The analysis reveals that as the hyperparameter γ grows, the overfitting decreases, allowing a smaller regularization factor α to maintain predictor stability. The most balanced trade-off is obtained at $\alpha = 1$ and $\gamma = 20$. Although these results (0.055644 for $c = 1$, with $m = 3200$) present an enhancement over those achieved through basic ridge regression (expected accuracy of 0.052104 for $c = 1$, obtained with $m = 66780$), it is evident that with a sample size of $m = 3200$, the model remains overly simplistic. This high bias suggests that the model is still insufficiently complex to effectively capture the intricacies of the phenomenon being studied.

These results could potentially illustrate the 'no free lunch' theorem: not every theoretically consistent algorithm can produce a predictor converging to the Bayes optimal predictor as m increases (especially for large but finite sizes m). This suggests that for complex learning tasks $(\mathcal{D}, \ell)$, even with a very large m, the accuracy of the predictor generated by an algorithm like the Kernel Ridge Regression might remain low (small variance but still high bias). However, we can't definitively confirm this hypothesis because, although the achieved accuracy is still notably close to 0 and the sample size $m = 3200$ is relatively small compared to the maximum available size $m = 66780$, the spatial and temporal complexity of the problem precludes the generation of a learning curve that fully captures how the risk of the optimal predictor generated by the kernel ridge regression evolves as m increases.

Anyway, it is possible to slightly improve the accuracy of the optimal predictor while maintaining the training set size at approximately $m' \approx 3200$. This can be achieved by using the divide and conquer technique explained earlier in section 3.1. Table 2 reports, for the $c = 1$ problem, the results of

M	m'	Gamma	Alpha	Max. training acc. estimate	Max. test acc. estimate
20	3200	10	5	0.110672	0.110108
20	3200	20	1	0.115148	0.114499
20	3200	20	1	0.115148	0.114499

Table 2: Continuous features ($c = 1$). 5-fold best CV estimate with A =DCKRR and $\Theta_0 = [(\alpha_0, \gamma_0), (\alpha_1, \gamma_1)] = [(5, 10), (1, 20)]$.

the 5-fold best CV estimate run on the DCKRR algorithm, considering the hyperparameter space $\Theta_0 :=: [(\alpha_0, \gamma_0), (\alpha_1, \gamma_1)] =: [(5, 10), (1, 20)]$ (chosen based on the results obtained in Table 1) and $M = 20$ (subsets into which the training set is divided). The results show that the accuracy of the average predictor is slightly more than double (expected risk equal to 0.114499 for $\theta = (1, 20)$) the accuracy of the predictor obtained with simple kernel ridge regression.

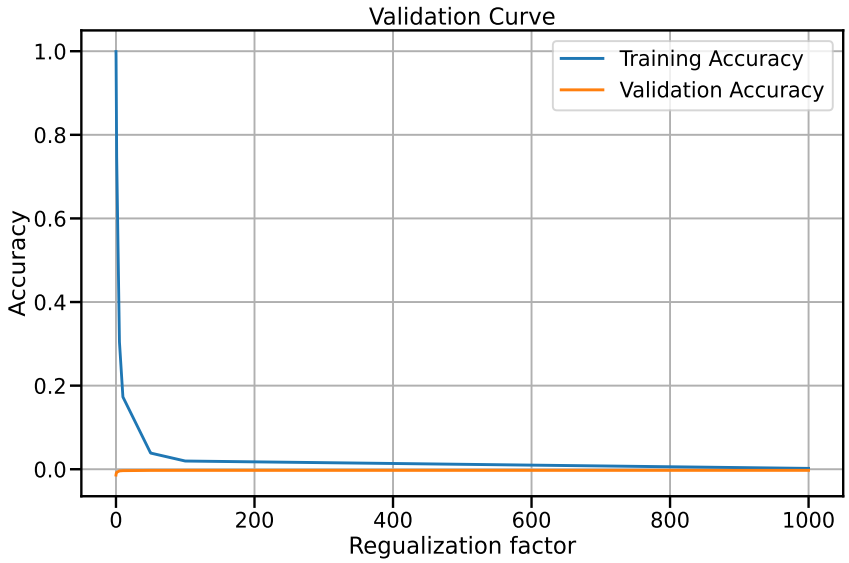

Figure 1: Validation curve $\alpha = 1000$ e $\gamma = 0.00001$ ($c = 1$).

4.2 Continuous, ordinal (qualitative), binary and categorical features

In the final section, we explore kernel ridge regression designed specifically for the $S(c = 2)_r$ dataset. This dataset includes 157193 original samples with a mixture of category, ordinal, and binary properties. Due to limitations, we create the ridge regression model using a subset of only 4,000 samples ($m = 3200$), much like the $c = 1$ example. As for problem $c = 1$, we are forced here to consider only $r = 4000$ ($m = 3200$) samples to build the ridge regression predictor.

Gamma	Alpha	Max. training acc. estimate	Max. test acc. estimate
0.00001	1.0	0.998719	-0.000240
0.00100	1.0	0.998411	0.000774
0.01000	0.5	0.998353	0.002127
0.10000	0.1	0.998346	0.009407
1.00000	0.1	0.998306	0.079082
2.00000	0.1	0.947981	0.168577
4.00000	1.0	0.841538	0.232828
5.00000	5.0	0.795044	0.238412
10.00000	5.0	0.646683	0.278079
15.00000	5.0	0.573672	0.285697
20.00000	5.0	0.530968	0.284819
50.00000	1.0	0.442073	0.275469
100.00000	0.5	0.410365	0.251253
500.00000	0.1	0.360595	0.219017
4000.00000	0.1	0.338284	0.215640
15.00000	5.0	0.573672	0.285697

Table 3: Continuous, binary, ordinal and categorical features ($c = 2$). 5-fold best CV estimate, with alpha being chosen within $[\alpha_0, ..., \alpha_8] = [0.00001, 0.001, 0.01, 0.1, 1, 2, 4, 5, 10, 15, 20, 50, 100, 500, 4000]$.

However, we will eventually attempt to increase the kernel model's accuracy by applying the divide-and-conquer strategy to higher values of m' (by taking advantage of the processing power offered by Google's colab).

Although the one-hot-encoding strategy has shown to be effective in improving the ridge regression predictor's accuracy (as mentioned in the initial attached study), it introduces an excessive amount of new binary features. This substantial increase significantly impacts the runtime for both KRR and DCKRR. Therefore, out of the initial set of suggested encoding combinations, we chose the first one, i.e. the one that results in the fewest new attributes: "target encoding" for "track_genre" and "count encoding" for "artists," "album_name," and "track_name."

The outcomes of the 5-fold best CV estimate are shown in TABLE 3. Similar to problem $c = 1$, we use the hyperparameter space 'hyper_space' for analysis. Notably, as γ increases, the optimal predictor becomes more stable, allowing α to decrease. However, excessive growth in γ leads the outputted predictor to inadequately approximate the feature-target relationship, resulting in increased bias. Even with a modest sample size of only $m = 3200$, we achieve an estimated expected accuracy that outperforms the accuracy attained when describing X as in $c = 1$, specifically for $\gamma = 15$ and $\alpha = 5$. This performance almost matches that of classical ridge regression conducted on the same encoded features (which yielded an accuracy of 0.395789). However, the resultant predictor $f^*(\cdot)$ fails to deliver a satisfactory performance, which is not surprising given the sample size of $m = 3200$ that is undeniably too small.

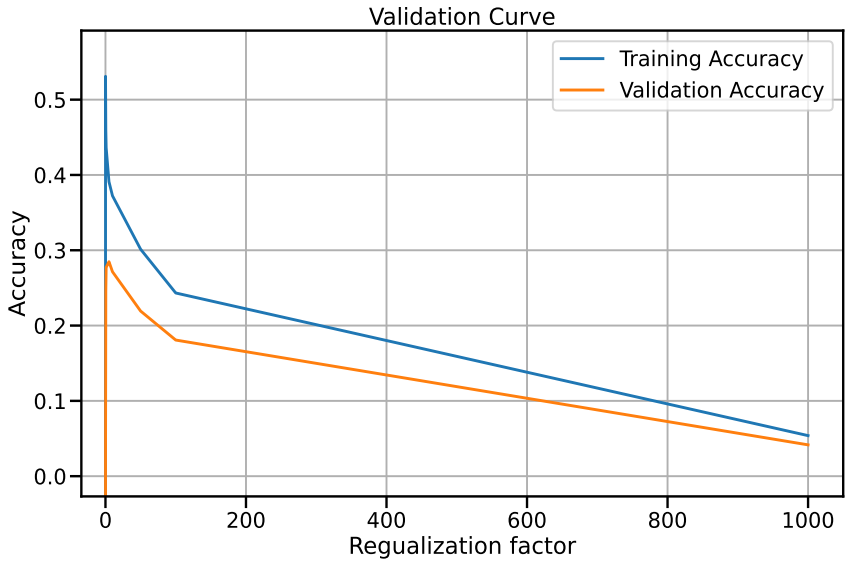

Figure 2: Validation curve $\alpha = 5$ e $\gamma = 15$ ($c = 2$).

The only way we can significantly increase the accuracy of $f^*(\cdot)$ is by employing the divide-and-conquer strategy. TABLE 4 shows that if we divide the entire dataset $S_r^{(c=2)}$ (with $r = 157193$) into M subsets, so that each training part S'_m ($S_m^{(-i)}$ in the cross validation context) has sample size equal to m' (which approximately takes increasing values 3200, 7000 and 10000), we witness a near doubling in accuracy, while maintaining the optimal choice of $\gamma = 15$ and $\alpha = 5$ for the hyperparameters. If it were feasible to further increase m', it would have been possible to achieve accuracy levels well beyond the

M	m'	Gamma	Alpha	Max. training acc. estimate	Max. test acc. estimate
36	3200	15	5	0.479499	0.477584
18	7000	15	5	0.49681	0.494555
11	10000	15	5	0.505889	0.503348

Table 4: Continuous, binary, ordinal and categorical features ($c = 2$). 5-fold best CV estimate with A =DCKRR (several sub-sizes m') and $\Theta_0 = [(\alpha_0, \gamma_0)] = [(5, 15)]$.

performance of the random predictor (i.e., 0.5 accuracy). The trade-off, however, is that employing the DCKRR algorithm with $m' \gtrsim 10000$ becomes infeasible on a single machine. Additionally, running DCKRR on $m' = 3200$ using Google Colab's resources would require approximately 2 hours and 38 minutes. For $m' = 7000$, the duration extends to about 3 hours and 30 minutes, and for $m' = 10000$, it reaches around 4 hours and 20 minutes. Indeed, the feasible solution appears to be the implementation of a distributed version of the DCKRR algorithm using 'pyspark'. By partitioning the data into chunks and storing them in a distributed file system like 'Hadoop FS', the algorithm could be executed effectively, overcoming the limitations of running it on a single machine.

5 Citations and references

References

Foster, John T. 2023. 'numerical methods book' https://johnfoster.pge.utexas.edu/numerical-methods-book/LinearAlgebra_LU.html.

Gretton, Arthur. 2022. 'introduction to rkhs, and some simple kernel algorithms', lectures from the course 'reproducing kernel hilbert spaces in machine learning'. *University College London* http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/rkhscourse.html.

Rudin, Cynthia. 2012. 'Kernels', lectures from the course 'Prediction: Machine Learning And Statistics' https://ocw.mit.edu/courses/15-097-prediction-machine-learning-and-statistics-spring-2012/pages/syllabus/.

Shawe-Taylor, John & Nello Cristianini. 2004. *Properties of kernels* 47–84. Cambridge University Press. doi:10.1017/CBO9780511809682.004.

Song, Changyue. 2020. 'a story of basis and kernel', part 1 and 2 http://songcy.net/posts/story-of-basis-and-kernel-part-1/.

Zhang, Yuchen, John Duchi & Martin Wainwright. 2015. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research* 16(1). 3299–3340.