International Journal of Computer (IJC)

ISSN 2307-4523 (Print & Online)

https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

A Comprehensive Overview of Kernels in Machine Learning: Mathematical Foundations and Applications

Omar El Khatib[a]*, Nabeel Alkhatib[b]

[a, b]*Math. and Computer Science Dept., Loyola University New Orleans, New Orleans, 70118 USA*

[a]*Email: oelkhat@loyno.edu*

[b]*Email: nalkhati@my.loyno.edu*

Abstract

Kernels play a fundamental role in machine learning, enabling algorithms to operate efficiently and effectively in high-dimensional spaces. In this paper, we provide a comprehensive overview of regression kernels in machine learning, focusing on their mathematical foundations, properties, and practical applications. We begin with an introduction to the concept of regression kernels and their significance in machine learning. Then, we delve into the mathematical formulation of regression kernels, exploring Mercer's theorem and positive semi-definite (PSD) kernels. Next, we discuss popular kernel functions with their respective properties and applications. After that we apply regression kernel to the bike sharing demand dataset as a case study and compare the different kernel functions. Finally, we explore kernel limitations and current research trends and emerging directions in kernel-based learning, offering insights into the future potential of this powerful methodology. This work aims to serve as a resource for both researchers and practitioners seeking a thorough understanding of regression kernel-based approaches in machine learning.

Keywords: Kernel Applications; Mathematical Foundations of Kernels; Kernel Trick; Kernel Properties.

1. Introduction

Machine learning (ML) is a branch of artificial intelligence (AI) that focuses on learning from data to make predictions to unseen data. Traditionally, the theory and algorithms of machine learning and statistics have been very well developed for the linear case. Linear classifiers work best when the dataset classes are linearly separable [1]. However, real world data analysis problems often require nonlinear methods to detect the kind of dependencies that allow successful prediction of target of interest. For nonlinear decision boundary, one approach that allows us to still utilize linear classifiers is to apply a kernel [2].

Received: 10/2/2024
Accepted: 10/16/2024
Published: 12/26/2024

----

* Corresponding author.

Kernels play a fundamental role in machine learning, enabling algorithms to operate efficiently and effectively in high dimensional spaces. Kernels use the linear classifier to build non-linear decision boundaries by transferring the input dataset into high dimension space [1, 2, 3].

In this paper, we provide a comprehensive overview of kernels in machine learning, focusing on their mathematical foundations, properties, and practical applications. Section 2 review previous work and our study. Then section 3 introduces the concept of regression kernels. Next, we delve into the mathematical formulation of regression kernels in section 4. After that we explore Mercer's theorem and positive semi-definite (PSD) kernels in section 5. Next, we discuss popular kernel functions such as linear, polynomial, Gaussian (RBF), Laplacian and sigmoid kernels, along with their respective properties and applications in section 6. Then, we discuss applications of kernels in section 7. After that we showed a practical application of using linear regression kernels on the bike sharing demand dataset from Kaggle to find the minimum error and discuss the limitation of regression kernels in section 8. Finally, we conclude with a discussion on current research trends and future directions in kernel-based learning in section 9.

2. Previous Work

Mercer's theorem [4] established the conditions under which a kernel function can be expressed as an inner product in a high-dimensional space. This theorem is the foundation for kernel methods, ensuring that the kernel matrix is positive semi-definite and valid for use in optimization problems. This work relies on Mercer's theorem to justify the use of kernels like the RBF, polynomial, and Laplacian kernels.

Support vector machines (SVMs) [5] are one of the earliest and most influential applications of kernel methods. They use kernels to transform data into a high-dimensional space where a hyperplane can separate classes. This work builds on the kernel trick introduced in SVMs but applies it to regression problems using Kernel Ridge Regression (KRR).

Saunders and his colleagues [6] introduced KRR as a kernelized version of ridge regression. KRR has been applied to various domains, including bioinformatics, finance, and environmental modeling. Example: In bioinformatics, KRR has been used for protein structure prediction [7]. In this work applies KRR as a primary method to a novel domain—bike sharing demand prediction—demonstrating its versatility.

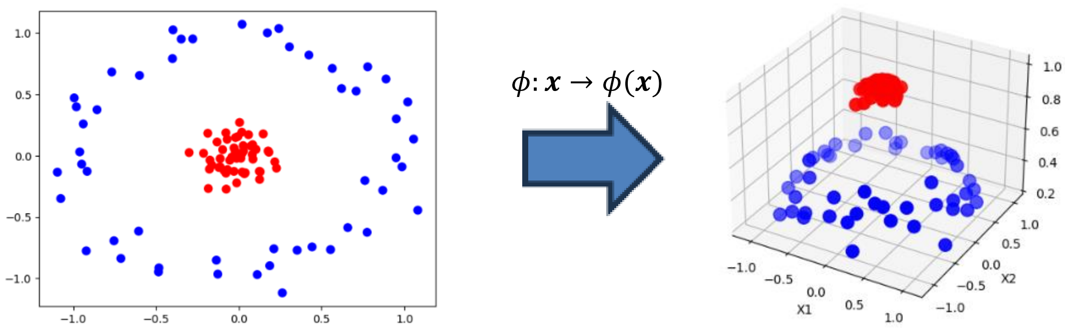
The choice of kernel function (e.g., RBF, polynomial, Laplacian) significantly impacts model performance. Schölkopf and Smola [8] discussed the properties of different kernels and their suitability for various tasks. Genton [9] provided a theoretical analysis of kernel functions, including the Laplacian kernel. This work compares multiple kernels and highlights the superior performance of the Laplacian kernel for bike sharing demand prediction. The performance of kernel methods depends heavily on hyperparameters like gamma, degree, and coef0.

Bergstra and Bengio [10] Introduced random search for hyperparameter optimization. Snoek and his colleagues [11] applied Bayesian optimization to hyperparameter tuning in kernel methods. This work uses grid search and cross-validation to tune hyperparameters, building on these methodologies.

In this paper, KRR is applied to the bike sharing demand dataset, a novel application that has not been extensively studied using kernel methods. This contributes to the growing body of research on kernel methods in urban planning and transportation. provide a detailed comparison of multiple kernels (linear, RBF, polynomial, Laplacian) and identify the Laplacian kernel as the best-performing one for this dataset. This adds to the literature on kernel selection by demonstrating the effectiveness of the Laplacian kernel in a real-world regression task. This work highlights the importance of hyperparameter tuning (e.g., gamma, degree, coef0) in achieving optimal performance with KRR. It also provides insights into the robustness of the Laplacian kernel to outliers and its ability to capture localized patterns in the data.

3. Kernels

Let **X** denote the original input features space. Using a feature map function $\boldsymbol{\phi}(\mathbf{X})$ that maps the original input features **X** to a possibly higher dimensional space $\boldsymbol{\phi}(\mathbf{X})$, such that data in this higher dimensional space become linearly separable. That function $\boldsymbol{\phi}$ is a non-linear transformation. After that, use the linear classifier (SVM, logistic regression, linear regression, etc.) to the high dimension transformed feature vector $\boldsymbol{\phi}(\mathbf{X})$ to construct a non-linear classifier in the original input features **X**. For example, assume the following dataset presented in Figure 1a, applying kernel function transformation as in Figure 1b, the data becomes linearly separable, and we can apply linear classifier.



(a) Original dataset

(b) dataset after applying kernel function transformation

Figure 1: Shows the effect of kernels function transformation on the dataset.

4. Mathematical foundation of Kernels

In this section, we delve into the mathematical formulation of kernels ridge regression (KRR), exploring Mercer's theorem and positive semi-definite (PSD) kernels.

4.1. Definitions

Let **R** be the set of real numbers. Given a dataset $\boldsymbol{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n}, \boldsymbol{x}_i \in \boldsymbol{R}^d$ called the input features of d-dimension, $\boldsymbol{y}_i \in \boldsymbol{R}$ is called the target. Apply a transformation $\boldsymbol{\phi}$ to every input feature $\boldsymbol{x}_i$, i.e. $\boldsymbol{\phi}: \boldsymbol{x}_i \rightarrow \boldsymbol{\phi}(\boldsymbol{x}_i)$, where $\boldsymbol{\phi(x_i) \in R^q}$, usually $\boldsymbol{q \gg d}$ because dimensions are added by $\boldsymbol{\phi}$ to capture the non-linear interaction among the original features. This new representation, $\boldsymbol{\phi(x_i)}$, is very expressive and allows for complicated non-linear decision boundaries - but the dimensionality is extremely high. This makes our algorithm unbearably (and quickly prohibitively) slow [12].

For example, Assume: $\boldsymbol{x_i^T = (x_1, x_2, x_3, \dots, x_d)}$, and define the kernel transformation that generate all polynomial terms $\boldsymbol{\phi(x_i)^T = (1, x_1, x_2, \dots, x_d, x_1x_2, \dots x_{d-1}x_d, x_1x_2x_3, \dots, x_1x_2x_3 \dots x_d)}$, the dimension of $\boldsymbol{x_i}$ is $\boldsymbol{d \times 1}$, but the dimension of $\boldsymbol{\phi(x_i)}$ is $\boldsymbol{2^d \times 1}$.

The advantage of kernel function is simple transformation, and the problem stays convex and well behaved, (i.e. you can still use the original gradient descent code, just with very high dimensional representation). Disadvantage: $\phi(x_i)$ might be very high dimensional.

4.2 Kernel trick

The kernel trick is a way to avoid the computation in high dimension space generated by $\phi(x_i)$ and overcome the slow algorithm of computing it. It also avoids computing the weights parameter w in the prediction function $h(x_i) = w^T\phi(x_i)$. It is called "trick" because it avoids computing transforming the data points x_i to the high dimensional space of $\phi(x_i)$ for making complex and non-linear classifications decision boundary.

4.3 Linear regression with kernel trick

Linear regression uses the linear prediction $h(x_i) = w^T x_i$ and the square loss function $l(w)$ in evaluating the prediction:

$$l(w) = \sum_{i=1}^{n} (w^T x_\mathrm{i} - y_\mathrm{i})^2 \qquad \dots (1)$$

Where $\mathrm{x_i}$ is of dimension $(d+1) \times 1$ including the bias and w is the weights parameters of dimension $(d+1) \times 1$. The gradient descent rule, with step-size (or learning rate) $\alpha > 0$, updates w over time as $w = w - \alpha\left(\frac{\partial l}{\partial w}\right)$, were, from (1):

$$\frac{\partial l}{\partial w} = \sum_{i=1}^{n} 2(w^T \mathrm{x_i} - \mathrm{y_i})\mathrm{x_i} \qquad \dots (2)$$

We can express w as a linear combination of the input feature $\mathrm{x_i}$ as: $w = \sum_{i=1}^{n} s_i \mathrm{x}_i$, were, $s_i \in R$ (real number).

To prove that we are going to use induction. Denote w^t as w at iteration t [12].

1) Base case: Since the loss function is a convex function of w, then the final solution of w that minimize the loss function is independent of the initial values of w when applying gradient descent. Therefore, we can

initialize $w^0 = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}$. For this initial choice of w^0 at iteration 0, the linear combination in $w^0 = \sum_{i=1}^{n} s_i^0 \mathrm{x_I}$ is trivial by setting $s_1^0 = s_2^0 = \cdots = s_n^0 = 0$.

2) Assume at iteration t, $w^t = \sum_{i=1}^{n} s_i^t \mathrm{x_I}$ is true, we need to show that $w^{t+1} = \sum_{i=1}^{n} s_i^{t+1} \mathrm{x_I}$ is true. We can show that from the update rule of w in the gradient descent algorithm as $w^{t+1} = w^t - \alpha\left(\frac{dl}{dw}\right)$

Using equation (2) for $\frac{dl}{dw}$:

$$w^{t+1} = w^t - \alpha \sum_{i=1}^{n} 2\left({w^t}^T \mathrm{x}_i - y_i\right)\mathrm{x}_i$$

Using the assumption $w^t = \sum_{i=1}^{n} s_i^t \mathrm{x_I}$, then rewrite the update rule for w^{t+1} as:

$$w^{t+1} = \sum_{i=1}^{n} s_i^t \mathrm{x_i} - \alpha \sum_{i=1}^{n} 2\left(\sum_{j=1}^{n} s_j^t \mathrm{x_j}^T\, \mathrm{x}_i - y_i\right)\mathrm{x}_i$$

Which we can rewrite as:

$$w^{t+1} = \sum_{i=1}^{n}\left(s_i^t \mathrm{x_i} - 2\alpha\left(\sum_{j=1}^{n} s_j^t \mathrm{x_j}^T\, \mathrm{x}_i - y_i\right)\mathrm{x}_i\right)$$

Taking $\mathrm{x_i}$ as a common factor:

$$w^{t+1} = \sum_{i=1}^{n}\left(\left(s_i^t - 2\alpha\left(\sum_{j=1}^{n} s_j^t \mathrm{x_j}^T\, \mathrm{x}_i - y_i\right)\right)\mathrm{x}_i\right)$$

Finally, setting $s_i^{t+1} = s_i^t - 2\alpha\left(\sum_{j=1}^{n} s_j^t x_j^T x_i - y_i\right)$, then we have:

$$w^{t+1} = \sum_{i=1}^{n} s_i^{t+1} \mathrm{x}_i \qquad \blacksquare$$

This completes the proof.

From the previous proof, the update rule for s is:

$$s_i^{t+1} = s_i^t - 2\alpha\left(\sum_{j=1}^{n} s_j^t x_j^T x_i - y_i\right) \quad \ldots (3)$$

Rewrite the loss function $l(w)$ in terms of s by replacing w with $\sum_{i=1}^{n} s_i \mathrm{x}_i$ as follows:

$l(w) = \sum_{i=1}^{n}(w^T \mathrm{x}_i - y_i)^2$ From equation (1)

$$l(s) = \sum_{i=1}^{n}\left(\left(\sum_{j=1}^{n} s_j x_j\right)^T x_i - y_i\right)^2$$

$$l(s) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} s_j x_j^T x_i - y_i \right)^2 \quad \ldots (4)$$

The update rule for the s in every iteration t and for every data point i is from equation (3):

$s_k^t = s_k^{t-1} - 2\alpha \sum_{j=1}^{n} (s_j^t x_j^T x_i - y_i)$

In other words, we can perform the gradient descent update without expressing w explicitly. We need to keep track of the n-coefficient $s_1, \cdots, s_n$. In addition, we have the loss function $l(s)$ in terms of s.

During test time, we also need these coefficients $s_1, \cdots, s_n$to make a prediction of a test-input x_t, and we can write the entire classifier $h(\mathrm{x}_i)$ in terms of the inner-product between the test point and the training points:

$h(\mathrm{x_t}) = w^T \mathrm{x_t} = \sum_{j=1}^{n} s_j \mathrm{x}_j^T \mathrm{x_t}$

The gradient descent algorithm becomes as follows:

```
1)      Initialize $s_1 = s_2 = \cdots = s_n = 0$
2)      For k=1 to max_iterations:
a.      Initialize loss = 0
b.      For each training example $\mathrm{x}_i \in D$ ($D$ is dataset)
 i.     Update $s_i = s_i - 2\alpha\left(\sum_{j=1}^{n} s_j x_j^T x_i - y_i\right)$
 ii.    Compute the loss: $loss = loss + \left(\sum_{j=1}^{n} s_j \mathrm{x}_j^T \mathrm{x_i} - y_i\right)^2$
```

The inner product: $\mathrm{x_j^T} \mathrm{x}_i$ keep appearing in the above algorithm. If we replace $\mathrm{x_i}$ with $\Phi(\mathrm{x_i})$ that is very high dimensional space. Then we must replace the inner product $x_j^T \mathrm{x}_i$ with $\Phi(\mathrm{x_j})^T \Phi(\mathrm{x_i})$ in which requires very large memory space and computation resources.

If $\Phi(\mathrm{x})$ generate all polynomials, i.e. $\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ x_d x_d \\ x_1 x_2 x_3 \\ \vdots \\ x_1 x_2 x_3 \cdots x_d \end{bmatrix}$ with dimension $2^d \times 1$. Then $\Phi(\mathrm{x_j})^T \Phi(\mathrm{x_i})$ can be formulated as, rename variables x_i and x_j as x and z, respectively for simplicity of notations, each of dimension $d \times 1$: $\mathrm{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$ and $\mathrm{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix}$

The mapping $\phi(\mathrm{x})$ and $\phi(\mathrm{z})$ is: $\phi(\mathrm{x}) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1x_2 \\ \vdots \\ x_{d-1}x_d \\ x_1x_2x_3 \\ \vdots \\ x_1x_2x_3 \cdots x_d \end{bmatrix}$ and $\phi(\mathrm{z}) = \begin{bmatrix} 1 \\ z_1 \\ \vdots \\ z_d \\ z_1z_2 \\ \vdots \\ z_{d-1}z_d \\ z_1z_2z_3 \\ \vdots \\ z_1z_2z_3 \cdots z_d \end{bmatrix}$

Then, the inner product of $\phi(\mathrm{x})\phi(\mathrm{z})$ is:

$$\phi(\mathrm{x})\phi(\mathrm{z}) = 1 \cdot 1 + x_1z_1 + \cdots + x_dz_d + x_1x_2z_1z_2 + \cdots + x_1 \cdots x_dz_1 \cdots z_d$$

However, we can rewrite the inner product $\phi(\mathrm{x})\phi(\mathrm{z})$ as: $\phi(\mathrm{x})\phi(z) = \prod_{k=1}^{d}(1 + x_kz_k)$ [4]

We can prove this using mathematical induction:

1) Base case, if d=2 then we have: $\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $\mathrm{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

Then: $\phi(\mathrm{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1x_2 \end{bmatrix}$ and $\phi(\mathrm{z}) = \begin{bmatrix} 1 \\ z_1 \\ z_2 \\ z_1z_2 \end{bmatrix}$

The inner product is: $\phi(\mathrm{x})\phi(\mathrm{z}) = 1 \cdot 1 + x_1z_1 + x_2z_2 + x_1x_2z_1z_2 = \prod_{k=1}^{2}(1 + x_kz_k)$

Since: $(1 + x_1z_1)(1 + x_2z_2) = 1 + x_1z_1 + x_2z_2 + x_1x_2z_1z_2$

2) Assume, x and z are of dimension $d \times 1$, and $\phi(\mathrm{x})\phi(\mathrm{z}) = \prod_{k=1}^{d}(1 + x_kz_k)$ Then we need to show that, if x and z are of dimension $(d + 1) \times 1$, then $\phi(\mathrm{x})\phi(\mathrm{z}) = \prod_{k=1}^{d+1}(1 + x_kz_k)$?

Expand $\phi(x)^T\phi(z)$:

$$\phi(\mathrm{x})\phi(\mathrm{z}) = 1 + x_1z_1 + \cdots + x_dz_d + x_{d+1}z_{d+1} + x_1x_2\, z_1z_2 + \cdots + x_{d-1}x_dz_{d-1}z_d + x_dx_{d+1}z_dz_{d+1} + \cdots + x_1x_2 \ldots x_dz_1z_2 \ldots z_d + x_1x_2 \ldots x_dx_{d+1}z_1z_2 \ldots z_dz_{d+1}$$

Expand the term: $\prod_{k=1}^{d+1}(1 + x_kz_k)$:

$$\prod_{k=1}^{d+1}(1 + x_kz_k) = \prod_{k=1}^{d}(1 + x_kz_k)\,(1 + x_{d+1}z_{d+1})$$

From the assumption:

$$\prod_{k=1}^{d}(1 + x_kz_k) = 1 + x_1z_1 + \cdots + x_dz_d + x_1x_2z_1z_2 + \cdots + x_{d-1}x_dz_{d-1}z_d + x_1x_2x_3z_1z_2z_3 + \cdots + x_1x_2 \ldots x_dz_1z_2 \ldots z_d$$

$\prod_{k=1}^{d+1}(1+x_k z_k)=\prod_{k=1}^{d}(1+x_k z_k)\,(1+x_{d+1}z_{d+1})$

$=(1+\mathrm{x}_1 z_1+\cdots+\mathrm{x_d z_d}+\mathrm{x_1 x_2}\ \mathrm{z_1 z_2}+\cdots+\mathrm{x_{d-1}x_d z_{d-1}z_d}+\cdots+\mathrm{x_1 x_2}\ldots\mathrm{x_d}z_1 z_2\ldots\mathrm{z_d})(1+\mathrm{x}_{d+1}\mathrm{z}_{d+1})$

$=1+x_1 z_1+\cdots x_{d+1}z_{d+1}+x_1 x_2 z_1 z_2+\cdots x_d x_{d+1}z_d z_{d+1}+\cdots+x_1 x_2\ldots x_{d+1}z_1 z_2\ldots z_{d+1}$

This last term matches: $\phi(x)^T\phi(z)$. ■

This completes the proof.

The sum of 2^d terms become the product of d terms. We can compute the inner-product from the above formula in time O(d) instead of O(2^d). Therefore, we define the function:

$$k(\mathrm{x}_i,\mathrm{x}_j)=\Phi(\mathrm{x}_i)\Phi(\mathrm{x}_j)$$

Where $k(\mathrm{x}_i,\mathrm{x}_j)$ is called kernel function. With finite training set of n samples, the inner products are often pre-computed and stored in a Kernel matrix $K_{ij}=\phi(\mathrm{x}_i)\phi(\mathrm{x_j})$.

If we store the matrix K, we only need to do simple inner-product lookups and low dimensional computations throughout the gradient descent algorithm. The final classifier becomes:$h(\mathrm{x_t})=\sum_{j=1}^{n}s_j\,k(\mathrm{x_j},\mathrm{x_t})$.

During training in new high dimension space of ϕ(x) we want to compute s_i using the update rule in equation (3), rewritten here:

$s_i=s_i-2\alpha\left(\sum_{j=1}^{n}s_j x_j x_i-y_i\right)=s_i-2\alpha\left(\sum_{j=1}^{n}s_j K_{ij}-y_i\right)$

The loss function can be computed as: $loss=loss+\left(\left(\sum_{j=1}^{n}s_j\mathrm{K_{ij}}\right)-y_i\right)^2$

The amount of work per gradient update in the transformed space is O(n^2) far better than O(2^d).

The gradient descent algorithm revised is as follows:

```
1)    Initialize $s_1=s_2=\cdots=s_n=0$
2)    Compute $K_{ij}=x_i x_j$
3)    For k=1 to max_iterations:
a.    Initialize loss = 0
b.    For each training example $\mathrm{x}_i\in D$
  i.      Update $s_i=s_i-2\alpha\left(\sum_{j=1}^{n}s_j K_{ij}-y_i\right)$
  ii.     Compute the loss: $loss=loss+\left(\sum_{j=1}^{n}s_j K_{ij}-y_i\right)^2$
4)    Do predictions: $h(x_t)=\sum_{j=1}^{n}s_j\,k(\mathrm{x_j},\mathrm{x_t})$
```

4.4 Linear regression with L2-regularization (ridge regression kernel KRR)

Adding L2-regularization to the square loss function is $l(w) = \sum_{i=1}^{n}(w^T x_i - y)^2 + \lambda\|w\|_2^2$.

The derivative of the loss function with respect to *w* is:

$$\frac{\partial l}{\partial w} = \sum_{i=1}^{n} 2(w^T x_i - y_i)x_i + 2\lambda w$$

We can rewrite the loss and the derivative in terms of the coefficients $s_1, s_2, \cdots, s_n$ using $w = \sum_{i=1}^{n} s_i x_i$:

$$l(s) = \sum_{i=1}^{n}\left(\left(\sum_{j=1}^{n} s_j x_j\right)x_i - y_i\right)^2 + \lambda\left\|\sum_{j=1}^{n} s_j x_j\right\|_2^2$$

$$\frac{\partial l}{\partial w} = \sum_{i=1}^{n}\sum_{j=1}^{n} 2\left(s_j x_j x_i - y_i\right)x_i \; + 2\lambda\sum_{i=1}^{n} s_i x_i$$

$$\frac{\partial l}{\partial w} = 2\sum_{i=1}^{n}\sum_{j=1}^{n}\left(s_j x_j x_i - y_i\right)x_i \; + \lambda s_i x_i$$

Since we wrote the loss function and its derivate in terms of the coefficients $s_1, \cdots, s_n$ then we can proceed as before in the update rule of the coefficient $s_1, \cdots, s_n$:

$$s_i = s_i - 2\alpha\left(\left(\sum_{j=1}^{n}\left(s_j x_j x_i - y_i\right)\right) + \lambda s_i x_i\right)$$

The prediction is the same as before: $h(x_t) = \sum_{j=1}^{n} s_j\, k\left(x_j, x_t\right)$

5. Properties of Kernel functions

The kernel matrix K must be symmetric [1, 2], i.e. $K_{ij} = K_{ji}$ and matrix K is positive semi-definite [3, 4].

Definition: A matrix $A \in R^{n\times n}$ is positive semi-definite if and only if $\forall q \in R^n, q^T A q \geq 0$.

We can show that the kernel matrix K is symmetric since $K_{ij} = k\left(x_I, x_j\right) = \phi(x_i)\cdot\phi\left(x_j\right) = \phi(x_i)^T\phi\left(x_j\right) = \phi\left(x_j\right)^T\phi(x_i) = \phi\left(x_j\right)\cdot\phi(x_i) = k\left(x_j, x_i\right) = K_{ji}$.

To prove the kernel matrix *K* is a positive semi-definite then we need to show that $q^T K q \geq 0$ for any vector q.

Let $\phi_k(x)$ denote the kth coordinate of the vector $\phi(x)$, we find that for any vector q, we have:

$$\begin{aligned} q^T K q &= \sum_i\sum_j q_i K_{ij} q_j \\ &= \sum_i\sum_j q_i \phi(x_i)^T\phi\left(x_j\right) q_j \\ &= \sum_i\sum_j q_i \sum_k \phi_k(x_i)\phi_k\left(x_j\right) q_j \end{aligned}$$

$$= \sum_i \sum_j \sum_k q_i \phi_k(x_i) \phi_k(x_j) q_j$$

$$= \sum_k \sum_i \sum_j q_i \phi_k(x_i) \phi_k(x_j) q_j$$

$$= \sum_k (\sum_i q_i \phi_k(x_i))^2 \geq 0$$

The second-to-last step uses the fact that $\sum_i \sum_j a_i a_j = (\sum_i a_i)^2$ for $a_i = q_i \phi_k(x_i)$. Since q was arbitrary, then it follows that K is a positive semi-definite matrix.

Hence, we have shown that if K is a kernel matrix for some feature mapping ϕ, then the corresponding kernel matrix $K \in R^{n \times n}$ is symmetric positive semi-definite matrix.

The kernel matrix K that is symmetric and positive semi-definite is called Mercer kernel due to Mercer theorem:

Theorem (Mercer). Let $K: R^d \times R^d \rightarrow R$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x_1, \cdots, x_n\}, n < \infty$, then the corresponding kernel matrix K is symmetric positive semi-definite matrix.

6. Popular Kernels

Some of the popular kernels:

1) Linear*:* $k(x, z) = x \cdot z$ this is just the linear regression with L2 regularization, but it can be faster to use a kernel matrix if the dimension d of the input features is high.
2) Polynomial $k(x, z) = (1 + \gamma x \cdot z)^d$ that depends on the polynomial degree d and gamma γ.
3) Radial Basis Function (RBF) (aka Gaussian Kernel): $k(x, z) = e^{-\frac{\|x-z\|^2}{2\sigma^2}} = e^{-\gamma \|x-z\|^2}$. The RBF kernel is the most popular Kernel; it is a universal approximator.
4) Laplacian kernel: $k(x, z) = e^{-\gamma \|x-z\|}$
5) Sigmoid kernel: $k(x, z) = tanh(\gamma x^T z + c)$

To show that the Radia Basis Function (RBF) corresponds to a kernel matrix K; we need to show that K is symmetric and positive semi-definite.

Symmetric: $K_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} = e^{-\frac{\|x_j - x_i\|^2}{2\sigma^2}} = K_{ji}$ (trivial case).

Positive semi-definite: we need to show that $q^T K q \geq 0$

$$q^T K q = \sum_i \sum_j q_i K_{ij} q_j$$

$$= \sum_i \sum_j q_i e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} q_j$$

$$= \sum_i \sum_j q_i q_j e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

$$= \sum_i \sum_j q_i q_j \sqrt{e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}} \sqrt{e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}}$$

$$= \sum_i \sum_j \left(q_i \sqrt{e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}} \right) \left(q_j \sqrt{e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}} \right)$$

$$= \sum_i \left(q_i \sqrt{e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}} \right)^2 \geq 0$$

So, indeed the RBF kernel matrix K is a positive semi-definite matrix. Therefore, the RBF is a valid kernel function. Similarly, we can show Exponential kernel matrix and Laplacian kernel matrix are symmetric and positive semi-definite matrices.

For Sigmoid kernel function $tanh(\gamma x^T z + c)$, we can show that the matrix K corresponds to Sigmoid kernel function is symmetric and positive semi-definite matrix as follows:

Symmetric: $K_{ij} = tanh\left(\gamma x_i^T x_j + c\right) = K_{ji} = tanh\left(\gamma x_j^T x_i + c\right)$

Positive semi-definite matrix:

$$q^T K q = \sum_i \sum_j q_i q_j K_{ij}$$

$$= \sum_i \sum_j q_i q_j \, tanh\left(\gamma x_i^T x_j + c\right)$$

$$= \sum_i \sum_j q_i q_j \left(\sqrt{tanh\left(\gamma x_i^T x_j + c\right)} \right) \left(\sqrt{tanh\left(\gamma x_i^T x_j + c\right)} \right)$$

$$= \sum_i \sum_j \left(q_i \sqrt{tanh\left(\gamma x_i^T x_j + c\right)} \right) \left(q_j \sqrt{tanh\left(\gamma x_i^T x_j + c\right)} \right)$$

$$= \sum_i \sum_j \left(q_i \sqrt{tanh\left(\gamma x_i^T x_j + c\right)} \right)^2 \geq 0$$

So, indeed the Sigmoid kernel matrix K is positive semi-definite matrix. Therefore, the Sigmoid function is a valid kernel function.

7. Applications for Kernels

Kernels find applications across various machine learning algorithms, including:

- Support Vector Machines (SVMs): SVMs utilize kernels to map data into high-dimensional spaces, enabling the learning of nonlinear decision boundaries [13, 14, 15, 16, 17, 18].
- Kernel Ridge Regression: In kernel ridge regression, kernels are used to regularize the regression problem in high-dimensional spaces, preventing overfitting [13, 16, 17, 18].
- Kernel Principal Component Analysis (PCA): Kernels are employed in kernel PCA to perform nonlinear dimensionality reduction, preserving complex structures in the data [18, 19].
- Kernelized deep learning which uses linear activations to account for non-linearity by the kernel trick [13, 21, 22, 23, 24].
- Kernelized clustering: Kernel methods can be applied to clustering algorithms such as kernel k-means or spectral clustering. By employing kernel functions, these algorithms can effectively cluster data points in nonlinearly separable spaces [25].

8. Experimental Results

The library sklearn in Python provides kernel ridge regression (KRR) [26], it combines the linear regression with kernel trick that uses square loss and L2-norm regularization. The use of kernel trick means it handles nonlinear data without the need for explicit transformation into a higher-dimensional space. Kernel trick is a mathematical kernel function that computes the dot products in the high-dimensional feature space without explicitly mapping the data into that space. The following is the syntax for the linear regression kernel in sklearn:

class sklearn.kernel_ridge.**KernelRidge** (alpha=1, *, kernel = 'linear', gamma = None, degreee = 3, coef0=1)

where alpha (α) is the regularize coefficient (λ in the section 2), kernel can be 'linear', poly' or 'polynomial', 'rbf', 'laplacian', 'sigmoid'. The parameter gamma is $\frac{1}{2\sigma^2}$ in 'rbf', sigmoid, polynomial and Laplacian. In Kernel Ridge Regression, the parameter gamma plays a critical role in defining the behavior of kernel functions. High values of gamma lead to more complex models, while lower gamma results in smoother, more generalized models. The degree parameter is used with polynomial kernel.

In addition, you can define your own kernel function by providing kernel call back function that will be applied to each pair of samples in the training dataset. The callable function should take two rows from input X and return the corresponding kernel value as a single number. Also, you can set the kernel as pre-computed which means that the input X is assumed to be a kernel matrix.

Applying Kernel Ridge from sklearn on bike sharing demand dataset from Kaggle competition (where the target variable is the number of rentals per day). We tried different kernels like linear kernels, polynomial kernels, RBF kernels, Laplacian kernels and user defined kernels.

8.1. Effect of certain parameters tuning

In Kernel Ridge Regression (KRR), the performance of the model depends heavily on the choice of the kernel function and its associated parameters (e.g., gamma, degree, coef0) for the bike sharing demand dataseet. In each case, choosing the best parameter that minimizes the RMSE for the test set as this indicates the best generalization performance, ensuring that the train RMSE is not too low at this point (to avoid overfitting).

8.1.1. Effect of polynomial degree in polynomial kernel

Low degree works well for linear or slightly non-linear relationships. For bike sharing demand dataset, a low degree polynomial might capture simple trends like linear increase in demand during weekends or holidays. On the other hand, high degrees work for highly non-linear relationships. For the bike sharing demand dataset, polynomial degree might capture complex interactions between features (e.g., temperature, humidity and seasonality). However, it might overfit if the degree is too high, leading to poor generalization. Figure 2 illustrates the effect of the polynomial degree on the root mean square error (RMSE).

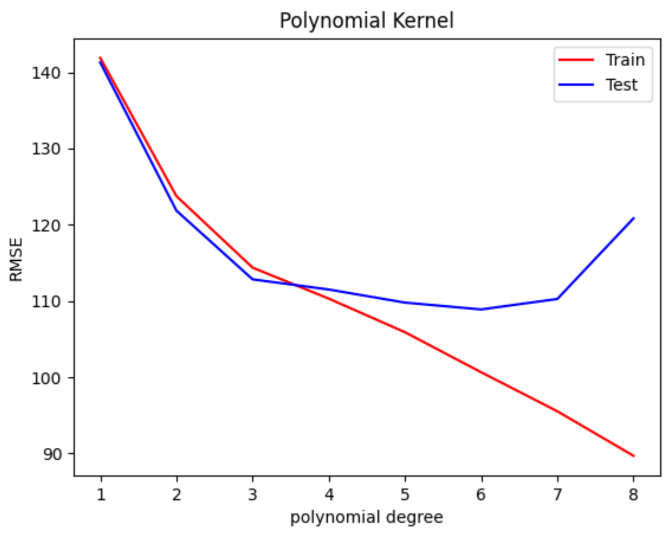


Figure 2: Effect of the polynomial degree on the model’s root mean square error.

Figure 2 shows the change in RMSE on the training and test set as the degree of the polynomial kernel increases, the optimal degree is the lowest RMSE that is at degree=4. A lower degree less than 4; the model is underfit while higher degree more than 4; the model overfit.

8.1.2. Effect of polynomial coef0 in polynomial kernel

The coefficient (bias) term in Polynomial kernel and Sigmoid Kernels is either zero or non-zero value. When the coefficient is zero, the model is simplified by removal of the bias term, this might work well if the data is centered around zero, but it could underfit if the data has a non-zero baseline. However, when the coefficient is non-zero, this adds flexibility to the model by introducing a bias term. For the bike sharing demand dataset, this might help capture the baseline demand levels. Figure 3 shows the train and test data when the bias term is added in the case of the bike sharing demand dataset. Figure 3 shows a line plot of the RMSE of the train and test set change as coefficient (bias) term varies. When the coefficient is zero, the error is high for train and test indicating that the model underfit. When the coefficient is non-zero, then the train and test errors are closer which allow the model to better fit the data. The best bias term (coef0) is at 1.

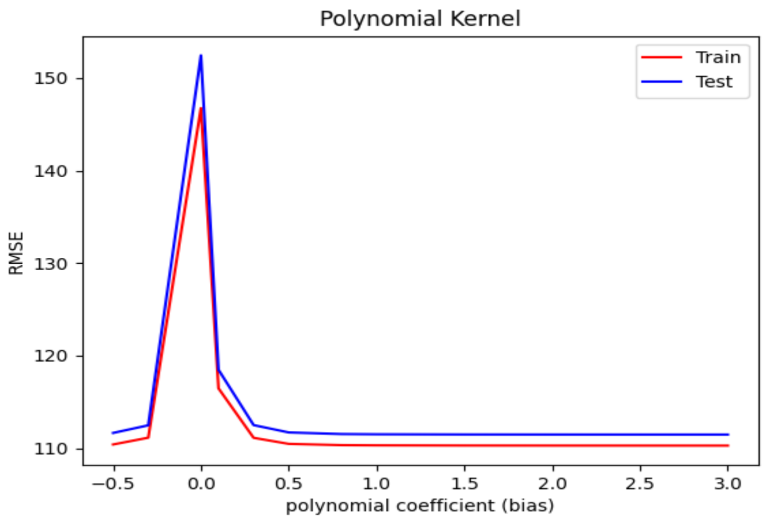


Figure 3: Effect of the coefficient (bias) term on the model accuracy, for polynomial kernel with degree=4.

8.1.3. Effect of gamma on model error

Gamma determines the scale of the kernel ($\gamma(x \cdot z)$), hence small gamma means a larger radius of influence for each training sample, so the kernel function decay slowly (smooth decay) that makes the model less sensitive to small fluctuations or noise in data; while large gamma means a smaller radius of influence, so the kernel function decays more quickly with the distance causing the model highly sensitive to small fluctuations or noise in the data. However, too small values of gamma cause the model to underfit, and too large gamma causes the model to overfit.

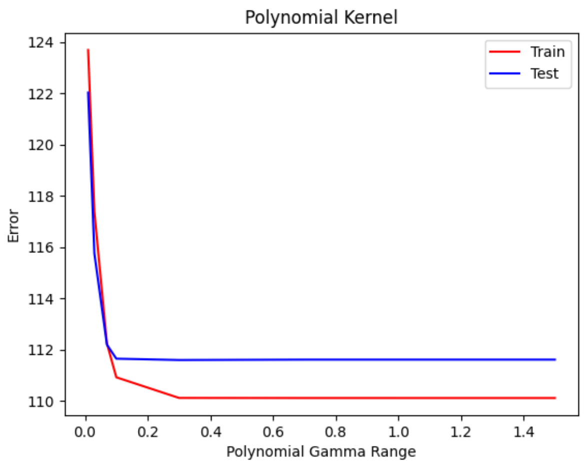


Figure 4: Effect of Gamma on polynomial kernel with degree d=4.

Figure 4 shows the effect of gamma on the polynomial kernel RMSE with degree 4 and coefficient of 1, when gamma is small (γ =0.01), the error was high, as gamma increases, the error decreases, the best gamma is in the range (0.1-1.5).

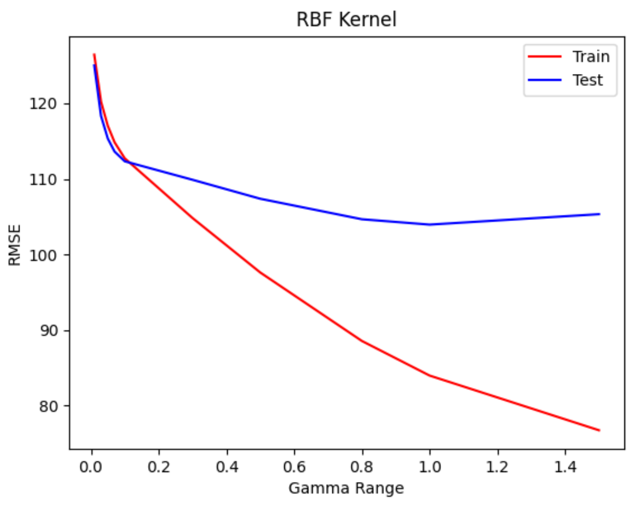


Figure 5: Effect of Gamma on RBF kernel

Figure 5 shows the effect of gamma on the RBF kernel RMSE on the train and test set, as gamma increases the model overfit. When the gamma is low, the model underfit. The best gamma is γ =0.1.

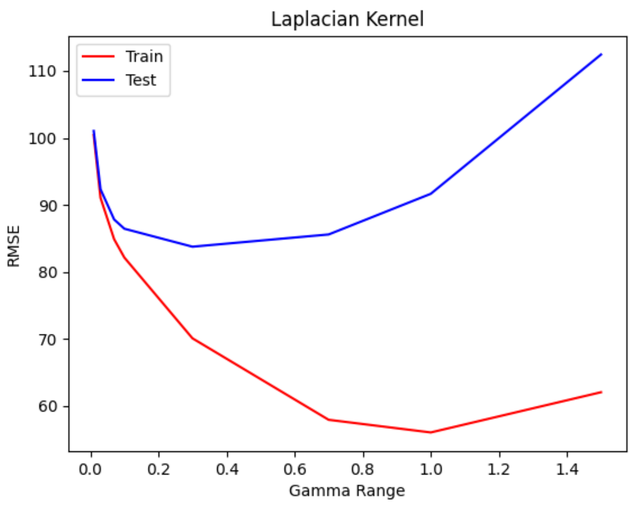


Figure 6: Effect of gamma on Laplacian kernel

Figure 6 shows the effect of gamma on the Laplacian kernel RMSE. As gamma increases the model overfit. As gamma is low, the model underfit. The best gamma is γ =0.03. The training error and test error dropped to 91.05 and 92.33, respectively; that is below the best RBF and Polynomial kernels.

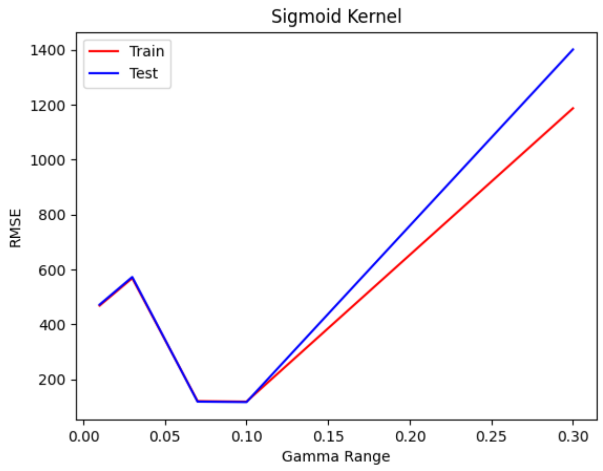


Figure 7: Effects of Gamma on Sigmoid Kernel.

Figure 7 shows the effect of gamma on Sigmoid kernel RMSE. As gamma increases the model overfit. As gamma is low, the model underfit. The best gamma is $\gamma = 0.1$.

8.1.4. Comparing different kernels

Figure 8 shows a bar chart of the train and the test RMSE for each kernel on the bike sharing demand. The Laplacian has the lowest root mean-square-error (RMSE), indicating that it generalizes better to the bike sharing demand dataset compared to the other kernels.

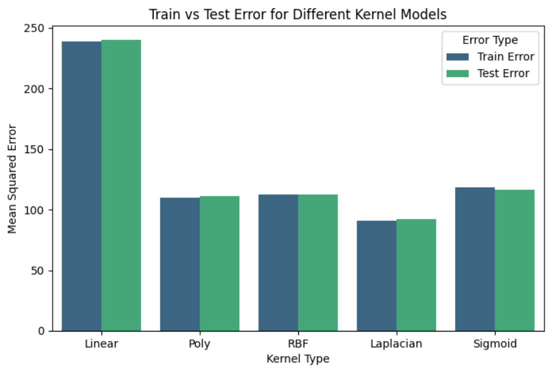


Figure 8: The bar chart shows the RMSE for train and test for each kernel with its best parameters for the bike sharing demand dataset.

Figure 9 shows a bar chart of the mean absolute error (MAE) for the train and test sets for each kernel on the bike sharing demand. The Laplacian has the lowest mean-absolute-error (MAE), indicating that it generalizes better to the bike sharing demand dataset compared to the other kernels.

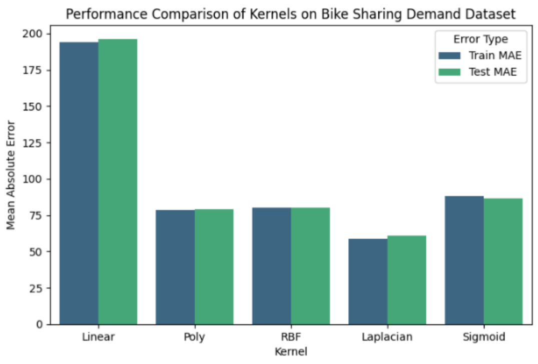


Figure 9: The bar chart shows the MAE for train and test for each kernel with its best parameters for the bike sharing demand dataset.

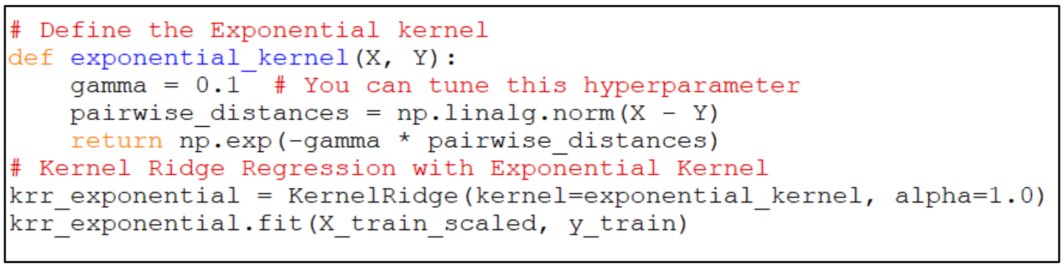
The Laplacian kernel uses the L1 norm (Manhattan distance), which is less sensitive to outliers compared to the L2 norm (Euclidean distance) used in the RBF kernel. Also, the Laplacian kernel decays more slowly than the RBF kernel, meaning it has a wider influence for each training example, this allows it to capture localized patterns in the data (e.g., sudden change in demand due to specific weather conditions) while still maintaining smoothness. In addition, Laplacian can model nonlinear relationships effectively, meaning it can capture the complex interactions between features like temperature, humidity and time of day in the bike sharing demand dataset.

Linear kernel performs poorly because the relationship between features in the bike sharing demand dataset is highly non-linear. The RBF kernel performs well but is lightly worse than the Laplacian kernel because it uses the L2 norm, making it more sensitive to outliers and tends to overfit if gamma is too large. The polynomial kernel performs well but is less flexible than the Laplacian kernel and requires careful tuning the degree and coefficient (bias) term, but it can overfit when the degree is too high.

In the bike sharing dataset, outliers might occur due to sudden spikes or drops in demand (e.g., during extreme weather events or holidays). The Laplacian kernel handles these outliers better, leading to more robust predictions.

Table 1 is a comparison between different kernel regression models with their strengths and weaknesses.

You can also define your own kernel using the sklearn. Figure 10 shows a code snippet of defining an exponential kernel that is not built-in sklearn library. Using the exponential kernel returns RMSE 105.67 for train and 111.30 for test.



```
# Define the Exponential kernel
def exponential_kernel(X, Y):
    gamma = 0.1  # You can tune this hyperparameter
    pairwise_distances = np.linalg.norm(X - Y)
    return np.exp(-gamma * pairwise_distances)
# Kernel Ridge Regression with Exponential Kernel
krr_exponential = KernelRidge(kernel=exponential_kernel, alpha=1.0)
krr_exponential.fit(X_train_scaled, y_train)
```

Figure 10: Code snippet for defining exponential kernel.

Table 1: Comparison summary between kernel regression models

Kernel model	Strengths	Weaknesses
Linear kernel	Simple and computationally efficient. Works well if the relationship between features and target is linear.	cannot capture nonlinear relationships. Likely to underfit the bike sharing dataset, which has complex interactions between features.
Polynomial kernel	Can capture polynomial relationships between features. Works well for modeling interactions between features (e.g., temperature and humidity).	Requires tuning of degree and coef0. Can overfit if the degree is too high.
RBF Kernel	Can capture nonlinear relationships and localized patterns. Works well for modeling smooth trends like seasonality and sudden changes due to weather.	Requires careful tuning of gamma to avoid overfitting or underfitting. Computationally more expensive than the linear kernel.
Sigmoid kernel	Can model S-shaped relationships. Might work well for certain types of nonlinear data.	Less commonly used for regression tasks like bike sharing demand prediction. Requires careful tuning of gamma and coef0

8.1.5. Predicted vs. actual plot

Figure 11 shows the predicted vs. actual plot that visualizes how well the best performing kernel that is Laplacian kernel captures the underlying patterns in the bike sharing demand dataset. This plot compares the model’s prediction against the true values, providing a clear visual representation of the model’s performance. The diagonal line represents the perfect prediction where the predicted value equals the actual value. The scatter points represent each point in the test set. The points are clustered closely around the diagonal line, which means that the model is accurately predicting the bike sharing demand. This indicates that the Laplacian kernel is effectively capturing the underlying patterns in the data.

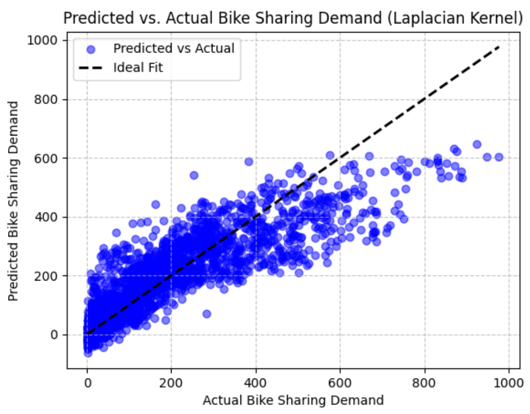


Figure 11: Plot of the predicated target vs the actual target for the bike sharing demand dataset using Laplacian kernel.

8.1.6. Heat map of RMSE Variance for Different Kernel Parameters

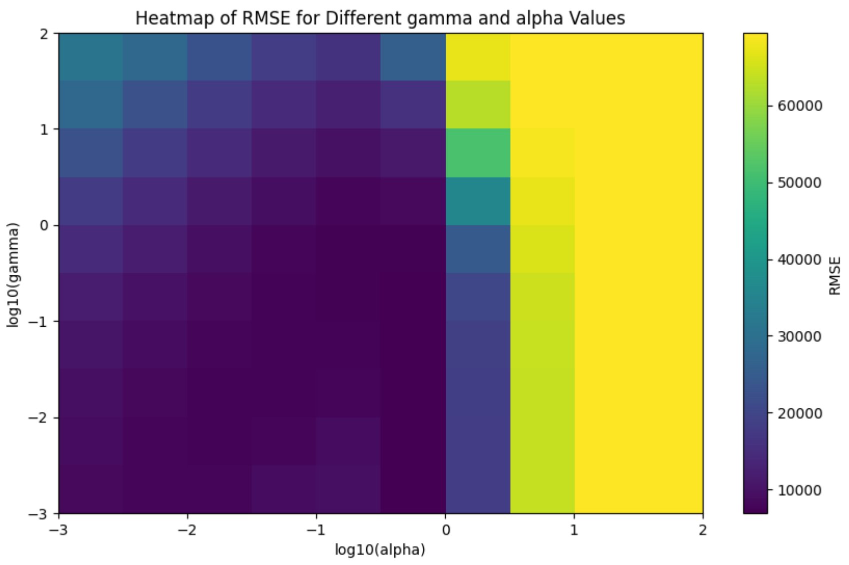


Figure 12: Heat map of how RMSE varies with different combinations of gamma and alpha in Laplacian kernel for bike sharing demand dataset.

Figure 12 shows the optimal values of gamma and alpha with the lowest RMSE in Laplacian kernel for the bike sharing demand dataset, this is indicated by the darkest color in the heatmap. The best values are Gamma=0.001 and alpha=0.001 which produces train RMSE of 88.02 and test RMSE of 90.96.

8.1.7. Learning curve

Learning curves are a powerful tool for diagnosing whether a model is overfitting or underfitting. They plot the training error and test error as a function of the number of training samples. By analyzing these curves, you can gain insights into the model's performance and identify potential issues.

Figure 13 shows the learning curve when using the optimal parameters for the Laplacian Kerner for the bike sharing demand dataset. Both the training RMSE and validation RMSE are low and converge as the number of training samples increases. A small gap indicates that the mode generalizes well to unseen data.

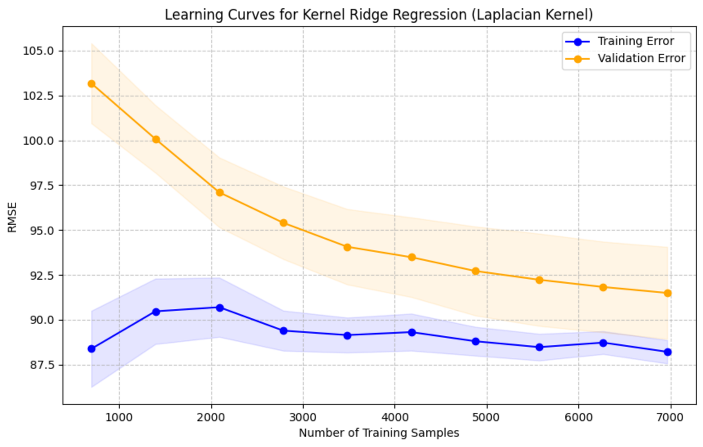


Figure 13: Learning curve for the Laplacian Kernel in the bike sharing demand dataset

8.2. Limitation of linear regression kernels in sklearn

Linear regression kernels in sklearn can be computationally expensive for large datasets due to the need to compute the kernel matrix. For the bike sharing demand dataset the shape of the dataset is (10886, 12); The bike sharing demand dataset may be limited in size, especially if it only covers a short time-period or a small geographic area. This works well for Kernel regression. However, trying kernel on bigger size datasets like the backpack price prediction from Kaggle; the data shape was (300000, 26) after converting categorical columns to one hot encoding, the kernel regression produces an insufficient memory because the kernel matrix is of size $300000 \times 300000 = 9 \times 10^{10}$. This limits the scalability of the approach to very large datasets or real-time applications. As a future work, explore scalable kernel approximations, such as the Nyström method [27, 28] or random Fourier features [27], to reduce computational complexity.

Kernel methods are highly sensitive to hyperparameters like gamma, degree, and coef0. Poorly tuned hyperparameters can lead to suboptimal performance, either underfitting or overfitting the data. An advanced method that automates hyperparameters tuning can be used such as Bayesian optimization or genetic algorithms to find the best hyperparameters more efficiently.

The performance of kernel methods depends heavily on the choice of kernel function (e.g., RBF, polynomial, Laplacian). An inappropriate kernel choice can lead to poor model performance. As future research, an investigation of adaptive kernel selection methods or ensemble approaches that combine multiple kernels to improve robustness.

Another limitation is missing data. Missing data can reduce the quality of the model's predictions and introduce uncertainty. Missing values must be handled with advanced imputation techniques before using kernel regression.

9. Conclusion

In conclusion, kernels play a crucial role in machine learning by enabling algorithms to operate effectively in high-dimensional spaces and capture complex patterns in data. By mapping data to higher-dimensional feature spaces, kernels allow algorithms like Support Vector Machines (SVMs) and Gaussian Processes (GPs) to achieve greater flexibility and accuracy. This has significantly expanded the application of kernel methods across a variety of tasks, reinforcing their value in modern machine learning.

The mathematical underpinnings of kernel methods, particularly the kernel trick, feature space transformations, and properties such as positive semi-definiteness, are core to their effectiveness. These mathematical concepts allow kernel methods to efficiently compute complex data relations without explicitly constructing high-dimensional feature spaces. This efficiency is vital for models working with non-linear data and provides a computationally feasible approach to problem-solving in diverse domains. Kernel methods applied across a broad range of domains, including image recognition, natural language processing, and bioinformatics.

Future research directions may include exploring novel kernel functions tailored to specific applications, developing efficient kernel methods for large-scale datasets, and investigating the theoretical properties of kernel-based learning algorithms. Despite the success of kernel methods, there are exciting areas for further exploration. Future research could focus on improving computational efficiency for large-scale datasets. Investigation of adaptive kernel selection methods or ensemble approaches that combine multiple kernels to improve robustness. Additionally, integrating kernel methods with deep learning offers new avenues for hybrid approaches. However, challenges remain, particularly with scalability in big data environments, which present ongoing research opportunities and motivate the development of advanced kernel techniques.

References

[1] Antoine Bordes, Seyda Ertekin, Jason Weston, Léon Bottou “Fast Kernel Classifiers with Online and Active Learning”. Journal of Machine Learning Research 6, pages 1579-1619, (2005).

[2] Bernhard Schölkopf, Alexander J. Smola, “Leaning with Kernels”. The MIT Press, 2002. ISBN 0-262-19475-9

[3] K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda and B. Scholkopf, "An introduction to kernel-based learning algorithms," in IEEE Transactions on Neural Networks, vol. 12, no. 2, pp. 181-201, March 2001

[4] Mercer, J., "Functions of positive and negative type and their connection with the theory of integral equations", *Philosophical Transactions of the Royal Society A*, **209** (441–458): 415–446, 1909.

[5] Cortes, C. and Vapnik, V., “Support-Vector Networks”. Machine Learning, 20, 273-297, 1995. http://dx.doi.org/10.1007/BF00994018.

[6] Saunders, C., Gammerman, A. and Vovk V., “Ridge Regression Learning Algorithm in Dual Variables”. ICML 98: Proceedings of the Fifteenh International Conference on Machine Learning. Pages 515-521, 1998.

[7] Schölkopf et al., “Kernel Methods in Computational Biology”. 2004 MIT Press (July 16, 2004)

[8] Schölkopf B. and Smola, A., “Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond”. The MIT Press, Cabridge Massachusetts, London, England. 2004.

[9] Genton, M. G., “Classes of Kernels for Machine Learning: A Statistics Perspective”. Journal of Machine Learning Research 2, 299-312, 2001.

[10] Bergstra, J. and BengioRandom, Y., “Search for Hyper-Parameter Optimization”. Journal of Machine Learning Research 13, 281-305, 2012.

[11] Snoek , J., Larochelle, H. and Adams, R. P. “Practical Bayesian Optimization of Machine Learning Algorithms”. Curran Associates, Inc.Vol 25, 2012.

[12] Weinberger., K., “Kernel Notes from Cornell CS4780 class”, Internet: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote13.html. Spring 2017

[13] Zareapoor, M., Shamsolmoali, P., Jain, D. K., Hoaxiang W. and Jie Y., “Kernelized support vector machine with deep learning: An efficient approach for extreme multiclass dataset”. Pattern Recognition Letterss, volume 115, pages 4-13, 2018.

[14] Herbrich, R., “Learning Kernel Classifiers Theory and Algorithms”. The MIT Press, 2001, ISBN: 9780262256339.

[15] Schölkopf, B., and Smola, A. J., “Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond”. MIT Press, 2002.

[16] Shawe-Taylor, J. and Cristianini, N. “Kernel Methods for Pattern Analysis”. Cambridge University Press, 2004.

[17] Hofmann, T., Scholkopf, B. and Smola, A. J., “A Review of Kernel Methods in Machine Learning”. Max Planck Institute for Biological Cybernetics, Technical report No. 156, 2006.

[18] Hofmann, T., Schölkopf. B., and Smola, A. J., "Kernel methods in machine learning" Annals of Statistics 36 (3) 1171 - 1220, June 2008.

[19] *Schölkopf, B., Smola, Alex; and Müller, Klaus-Robert, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". Neural Computation. 10 (5): 1299–1319, 1998. doi:10.1162/089976698300017467. S2CID 6674407.*

[20] *Schölkopf, B., Smola, Alex. and Müller, Klaus-Robert, Nonlinear Component Analysis as a Kernel Eigenvalue Problem (PDF) (Technical report). Max-Planck-Institut für biologische Kybernetik. 44, 1996.*

[21] Wilson A., Hu Zhiting, Salakhutdinov R. and Xing P. E., "Deep Kernel Learning". Published of the 19th International Conference on Artificial Intelligence and Statistics, 6 November 2015.

[22] Majumdar, A. "Kernelized Linear Autoencoder". Neural Process Lett 53, 1597–1614 (2021). https://doi.org/10.1007/s11063-021-10467-0

[23] Zheng, X., Ni, Z., Zhong X. and Luo Y., "Kernelized Deep Learning for Matrix Factorization Recommendation System Using Explicit and Implicit Information," in IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 1, pp. 1205-1216, Jan. 2024, doi: 10.1109/TNNLS.2022.3182942.

[24] Cho, Y. and Saul, L., "Kernel methods for deep learning". Advances in neural information processing systems, 2022.

[25] Wu, C., Khan, Z., Chang, Y., Ioannidis, S. and Dy, J., "Deep Kernel Learning for Clustering". Proceedings of the 2020 SIAM International Conference on Data Mining, pages 640-648, 2020.

[26] Murphy, K. P., "Machine Learning: A probabilistic Perspective". The MIT Press, Chapter 14.4.3, pp. 492-493, 2012.

[27] Ali Rahimi and Ben Recht, "Random Feature for Large-Scale-Kernel Machines". Advances in Neural Information Processing Systems, 2017.

[28] Williams, C. K. I. and Seeger, M, "Using the Nyström Method to Speed Up Kernel Machines". In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), Advances in Neural Information Processing Systems 13 (NIPS) (pp. 682-688), 2001.