THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Optimizing performance in Gaussian Processes

Michael Ciccotosto-Camp

Supervisor: Fred (Farbod) Roosta
Co-Supervisors: Andries Potgieter
Yan Zhao

Bachelor of Mathematics (Honours)
June 2022

The University of Queensland
School of Mathematics and Physics

Contents

Acknowledgements

I would like to deeply thank my supervisor Dr. Masoud Kamgarpour for his advice and all of his time spent with me. I consider myself lucky and am glad to have been his student for my honours year. I would also like to thank my co-supervisor Dr. Anna Puskás for the same reasons. A special thanks to Dr. Valentin Buciumas for his time spent teaching me while he was at The University of Queensland.

Symbols and Notation

Matrices are capitalized bold face letters while vectors are lowercase bold face letters.

Syntax	*Meaning*
$\triangleq$	An equality which acts as a statement
$\lvert\boldsymbol{A}\rvert$	The determinate of a matrix.
$\langle\cdot,\cdot\rangle_{\mathcal{H}}$	The inner product with respect to the Hilbert space $\mathcal{H}$, sometimes abbreviated as $\langle\cdot,\cdot\rangle$ if the Hilbert space is clear from context.
$\lVert\cdot\rVert_{\mathcal{V}}$	The norm of a vector with respect to the vector space $\mathcal{V}$, sometimes abbreviated as $\lVert\cdot\rVert$ if the vector space is clear from context.
$\boldsymbol{x}^\intercal, \boldsymbol{X}^\intercal$	The transpose operator.
$\boldsymbol{x}^*, \boldsymbol{X}^*$	The hermitian operator.
$\boldsymbol{a}.*\boldsymbol{b}$ or $\boldsymbol{A}.*\boldsymbol{B}$	Element-wise vector (matrix) multiplication, similar to Matlab.
$\propto$	Proportional to.
∇ or $\nabla_{\boldsymbol{f}}$	The partial derivative (with respect to $\boldsymbol{f}$).
∇	The Hessian.
$\sim$	Distributed according to, example $x \sim \mathcal{N}(0,1)$
$\mathbf{0}$ or $\mathbf{0}_n$ or $\mathbf{0}_{n\times m}$	The zero vector (matrix) of appropriate length (size) or the zero vector of length n or the zero matrix with dimensions $n \times m$.
$\mathbf{1}$ or $\mathbf{1}_n$ or $\mathbf{1}_{n\times m}$	The one vector (matrix) of appropriate length (size) or the one vector of length n or the one matrix with dimensions $n \times m$.
$\mathbb{1}_{n\times m}$	The matrix with ones along the diagonal and zeros on off diagonal elements.

$\boldsymbol{A}_{(\cdot,\cdot)}$	Index slicing to extract a submatrix from the elements of $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, similar to indexing slicing from the python and Matlab programming languages. Each parameter can receive a single value or a 'slice' consisting of a start and an end value separated by a semicolon. The first and second parameter describe what row and columns should be selected, respectively. A single value means that only values from the single specified row/column should be selected. A slice tells us that all rows/columns between the provided range should be selected. Additionally if now start and end values are specified in the slice then all rows/columns should be selected. For example, the slice $\boldsymbol{A}_{(1:3,j:j')}$ is the submatrix $\mathbb{R}^{3 \times (j'-j+1)}$ matrix containing the first three rows of $\boldsymbol{A}$ and columns j to j'. As another example, $\boldsymbol{A}_{(:,j)}$ is the j^{th} column of $\boldsymbol{A}$.
$\boldsymbol{A}^{\dagger}$	Denotes the unique psuedo inverse or Moore-Penore inverse of $\boldsymbol{A}$.
$\mathbb{C}$	The complex numbers.
C	The classes in a classification problem.
$\text{cholesky}\left(\boldsymbol{A}\right)$	A function to compute the Cholesky decomposition, $\boldsymbol{L}$, of the matrix $\boldsymbol{A}$, where $\boldsymbol{L}\boldsymbol{L}^{\intercal} = \boldsymbol{A}$.
$\text{cov}\left(\boldsymbol{f}\right)$	Gaussian process posterior covariance.
d	The number of features in the data set.
D	The dimension of the feature space of the feature mapping constructed in the Random Fourier Feature method.
$\mathcal{D}$	The dataset, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$.
$\text{diag}\left(\boldsymbol{w}\right)$	Vector argument, a diagonal matrix containing the elements of vector $\boldsymbol{w}$.
$\text{diag}\left(\boldsymbol{W}\right)$	Matrix argument, a vector containing the diagonal elements of the matrix $\boldsymbol{W}$.
$\mathbb{E}$ or $\mathbb{E}_{q(x)}\left[z(x)\right]$	Expectation, or expectation of $z(x)$ where $x \sim q(x)$.
$\mathcal{GP}$	Gaussian process $f \sim \mathcal{GP}\left(m(\boldsymbol{x}), k\left(\boldsymbol{x}, \boldsymbol{x}'\right)\right)$, the function f is distributed as a Guassian process with mean function $m\left(\boldsymbol{x}\right)$ and covariance function $k\left(\boldsymbol{x}, \boldsymbol{x}'\right)$.

$k(\cdot,\cdot)$	A covariance or kernel matrix.
$\text{l. s}\{\boldsymbol{x}_i\}_{i=1}^n$	The linear-spane of $\{\boldsymbol{x}_i\}_{i=1}^n$, that is, $\{\sum_{k=1}^n \lambda_k \boldsymbol{x}_k \mid \lambda_k \in \mathbb{R}\}$.
$\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'}$	For two data sets $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n]^\intercal \in \mathbb{R}^{n\times d}$ and $\boldsymbol{W}' = [\boldsymbol{w}'_1, \boldsymbol{w}'_2, \ldots, \boldsymbol{w}'_m]^\intercal \in \mathbb{R}^{n'\times d}$ the matrix $\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'} \in \mathbb{R}^{n\times n'}$ has elements $(\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'})_{i,j} = k\left(\boldsymbol{w}_i, \boldsymbol{w}'_j\right)$.
$\text{lin-solve}(\boldsymbol{A}, \boldsymbol{B})$	A function used to solve $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$ in the linear system $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$.
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ or $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$	(the variable $\boldsymbol{x}$ has a) Multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.
n and n_*	The number of training (and tests) cases.
N	The dimenion of the feature space.
$\mathbb{N}$	The natural numbers, $\mathbb{N} = \{1, 2, 3, \ldots\}$.
$\mathcal{O}(\cdot)$	Big-O notation. If a function $f \in \mathcal{O}(g)$ then the absolute value of $f(x)$ is at most a positive multiple of $g(x)$ for all sufficiently large values of x.
$y \mid x$ and $p(x \mid y)$	A conditonal random variable y given x and its probability density.
$\boldsymbol{Q}, \boldsymbol{V}$	Typically used to denote a matrix with orthonormal structure.
$\mathbb{R}$	The real numbers.
$\text{tr}(\boldsymbol{A})$	The trace of a matrix.
$\mathbb{V}$ or $\mathbb{V}_{q(x)}[z(x)]$	Variance, the variance of $z(x)$ when $x \sim q(x)$.
$\mathcal{X}$	Input space.
$\boldsymbol{X}$	The $n \times d$ matrix of training inputs.
$\boldsymbol{X}_*$	The $n_* \times d$ matrix of test inputs.
$\boldsymbol{x}_i$	The i^{th} training input.

$\mathbb{Z}$	The integers, $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$.

Introduction

Time series prediction (and related regressional tasks) is a subject of high interest across many disciplines of science and mathematics. The history of time series can be traced back to the birth of science in ancient Greece where Aristotle devised a systematic approach to weather forecasting in 350 BC in his famous treatise *Meteorologica*. This method was later used to help predict when certain meteorological induced events, such as the flooding of the Nile river [HHF73]. Statistical modelling for time series prediction would not come until the 20^{th} century where development of AutoRegressive Moving Average (ARMA) models which where first mentioned by Yule [Yul27] in 1927 and later popularized by Box and Jenkins in their book *Time Series Analysis* published in 1970 [Box08].

Given a data set of n observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where each input $x_i \in \mathbb{R}_{>0}$ is a time value and $y_i \in \mathbb{R}$ is a output or experimental observation that acts a function of time, the goal of time series prediction is to try and best predict a value $y_\star$ at a novel time $x_\star$. With computing power becoming ever more advanced and affordable, many have taken to Machine Learning (ML) to develop sophisticated models to address the problem of creating accurate yet computationally inexpensive time series predictors. Broadly speaking, ML is any class of heuristic algorithm that attempts to refine and develope some model to perform a "simple" task by learning through user provided input. ML is founded on the idea that any form of task learning is done through sensory input taken from the surrounding environment. More formally speaking, ML attempts to generate a function $f : X \to Y$, for some input set X and observation or output set Y, were the outputs given by f closely aligns to actual observations. It is tacitly assumed that the phenomena we are studying follows laws which admit mathematical formulation and that experimental results can be reproduced to some degree of accuracy. Typically, experiments will never produce exact values of the underpinning law, g. Instead experimental observations, y_i, will include a small amount of random error so that $y_i = g(x_i) + \varepsilon_i$ where $\varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \sigma^2\right)$.

A ML model will attempt to make accurate predictions using some simplified formulation of the world. The distribution corresponding to the probability of a prediction within the context of the "state of the world" is referred to as the *likelihood*. The uncertainty within the likelihood stems from the predictive limits of the model. These limitation usually arise as a consequence of selecting a model which is either too simple or complex. The "state of the world" is sometimes internally captured by the model as a set of mutable parameters $\boldsymbol{\theta}$. The process of taking observations and using them to form predictions is called *inference* which, in some sense, is synonymous with learning [VdW19].

ML can be applied to time series prediction in a fairly straight forward manner by simply teaching a ML algorithm the time series data set, $\mathcal{D}$, to hopefully produce a function f that serves as a good approximant for event prediction.

In this thesis we shall focus on a particular class of ML algorithms called Baysian models which, unsurprisingly, makes use of Bayesian statistics to drive inference. In Baysian models a *prior* distribution is used to quantify the uncertainty of the current state of the model before any observations are made. The model can then be updated once data is observed by using the likelihood to give a *posterior* distribution which represents the reduced uncertainty after "teaching" the model new observations. Methods of teaching a model how to change its behavior using a new set of observations often involves the use of a

loss function L. The loss function is used as an aid in deciding what action, a, should be taken in to best minimize uncertainty. The best action, roughly speaking, can be evaluated as

$$a_{\text{opt}} = \arg\min_{a} \int L\left(y_\star, a\right) p\left(y_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y}\right)\ dy_\star.$$

Interestingly, the best action does not rely so much on the model's internalized parameters but rather on the predictive distribution $p\left(y_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y}\right)$ [VdW19]. This key insight has spawned a class of ML algorithms that focuses on infering the function f directly by computing $p\left(f \mid \mathcal{D}\right)$ instead of finding optimal internal parameters using $p\left(\boldsymbol{\theta} \mid \mathcal{D}\right)$ [Mur12]. Models that perform inference in this manner are called *non-parameteric* models. Within the *non-parameteric* model paradigm, the predictive distribution can be represented as

$$p\left(y_\star \mid x_\star, \boldsymbol{X}, \boldsymbol{y}\right) = \int p\left(y_\star \mid f, x_\star\right) p\left(f \mid \boldsymbol{X}, \boldsymbol{y}\right)\ df$$

and once new data is observed the posterior can be updated using Baye's rule

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \qquad p\left(\boldsymbol{f}, f_\star \mid \boldsymbol{y}\right) = \frac{p\left(\boldsymbol{y} \mid \boldsymbol{f}\right) p\left(\boldsymbol{f}, f_\star\right)}{p\left(\boldsymbol{y}\right)}$$

[Ras06]. This thesis will focus on a particular non-parameteric Bayesian ML model called Gaussian processes (GPs). The over arching idea of GPs is to assign a prior probability to every possible function mapping from X to Y. While this does not appear to be computationally tractable due to the seemingly uncountable infinite number of mappings that would require checking, it turns out, these computations can infact be carried out given we are only seeking predictions at a finite number of points using a finite number of observations. GPs occupy a special place within the realm of ML since they account for uncertainty in a principled way, are relatively simple to implement and are highly modular allowing them to easily be incorporated into a larger systems. It is no surprise then that while other kernel methods (such as kernelized k^{th} nearest neighbors and ridge regression) are still overshadowed by their neural network cousins, GPs have made a quiet comeback in the ML community [Cao18].

The following example highlights a particular GP success story: a team of researchers led by Andries Potgieter at QAAFI (UQ) are currently investigating new digital approaches to accurately derive crop phenology stages (i.e. mid green, peak, flowering, grain filling and harvest) measured at field scale across large regions. Such methods can be used to better inform farmers and industry on the optimised time to plant various crops to minimize crop loss from environmental stresses such as frost and fungal disease. This involves analysing crop growth from previous seasons (i.e. 2018-2021) to forecast when certain phenological stages will take place in the current harvest. Outputs form this tool will allow producers to accurately map the temporal and spatial extend of phenology at a field and farm levels across different regions and seasons. This problem is readily converted into a time series problem. Originally, Potgieter's team surveyed a number of different parameteric models to carry out forecasting. However, the parameteric models we serverely limited in their ability to inform when key phenological stages would take place. After seeing the success of applying GPs to other remote sensing tasks [SD22] the team investigated the use of GPs in their own research to find that they could produce much higher resolution predictions from which they could infer a far richer phenological timeline [Pot13]. A comparision of using GPs over other parameteric models is shown in Figure 1. Potgieter's team found that the only draw back to using GPs was the lengthy run time required to create predictions and fears that

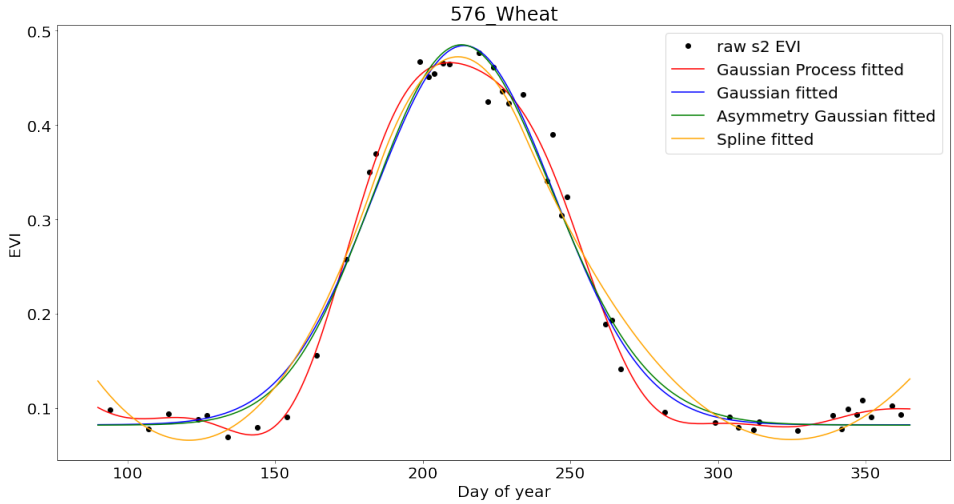


FIGURE 1. Potgieter's team found that GPs where superior in terms of predicting a phenological timeline for a number of common seasonal crops over other parameteric models.

collecting new data each season will only exacerbate the issue. This is a common problem shared by anyone wanting to use GPs. Due to their unwieldy $\mathcal{O}\left(n^3\right)$ runtime, where n is the number of observations, GPs become impractical to apply on datasets with $n > 10^5$ samples. As such, the goal of this thesis is to explore various avenues one can take to replace some of the more intense calculations of GPs with computationally more efficient approximations without overly sacrificing accuracy.

chapter **??** will give a more mathematical treatment of GPs starting from the ground through a review of some fundamental material from functional analysis also the theory behind the motivation of GPs before finally concluding with concrete algorithms for GP regression and classification. Chapters **??** and 1 will cover techniques for approximating a large matrix used with GPs that provides information on how similar each observation is to one other. chapter **??** then gives alternative methods for solving linear systems, an essential component required for the GP algorithm to work.

1. Random Fourier Features

As seen in **??** GPs rely heavily on the Gram matrix (see **??**) to create predictions based on training data $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ where $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^\intercal \in \mathbb{R}^{n \times d}$ and $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^\intercal \in \mathbb{R}^n$. Unfortunately, the size of the Gram matrix scales quadratically with the number of samples making it difficult to train using data sets with more than 10^5 samples. Instead the kernel function itself can be factorized allowing one to convert training and kernel evaluation into the corresponding operations of a linear machine by mapping data into a relatively low-dimensional randomized feature space. This idea was first introduced by Rahimi and Recht [Rah08] where they proposed that, instead of using a kernel function to implicitly lift data into a higher dimensional feature space, an explicit feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^D$ could be used to approximate k as $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle_{\mathbb{R}^N} \simeq \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle_{\mathbb{R}^D}$ where D is chosen so that $n \gg D$. Thus once $\varphi(\boldsymbol{x}_i)$ has been computed for each $\boldsymbol{x}_i$, every entry of the Gram matrix can be swiftly approximated as

$$\boldsymbol{K}_{ij} = \boldsymbol{K}_{ji} \simeq \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{y}_j) \rangle_{\mathbb{R}^D}.$$

Already there have been numerous applications of this technique in GPs that have seen improved time performance with little loss in prediction accuracy [Pot21].

1.1. Theory and Computation.

Contrary to the kernel trick, the Random Fourier Features (RFF) technique approximates $\langle \Phi(\cdot), \Phi(\cdot) \rangle_{\mathbb{R}^N}$ through an explicit feature mapping φ. The RFF technique hinges on Bochners theorem, stated without proof in Theorem 1, which characterises positive definite functions.

Theorem 1 (Bochner's)**.** *A continuous and shift-invariant function $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y}) = k(\Delta)$ is positive definite (see **??**) if and only if it can be represented as*

$$k(\boldsymbol{x} - \boldsymbol{y}) = \int_{\mathbb{C}^d} \exp\left(i \langle \boldsymbol{\omega}, \boldsymbol{x} - \boldsymbol{y} \rangle\right) \mu_k(d\boldsymbol{\omega})$$

where μ_k is a positive finite measure on the frequencies of $\boldsymbol{\omega}$ [Hah33, Liu21].

The spectral distribution μ_k can be represented as finite measure induced by the Fourier transformation. Choosing a kernel for which $k(\boldsymbol{0}) = 1$ normalizes μ_k to a probability distribution $p(\cdot)$. For instance, the spectral distribution of the Gaussian RBF kernel is

$$p(\boldsymbol{w}) = \frac{1}{\sqrt{(2\pi)^D \left|\frac{\sigma^2}{2} \mathbb{1}_{D \times D}\right|}} \exp\left(-\frac{1}{2} \boldsymbol{w}^\intercal \left(\frac{\sigma^2}{2} \mathbb{1}_{D \times D}\right)^{-1} \boldsymbol{w}\right) \tag{1}$$

[Rah08, page 3]. One caveat in Bochner's theorem is that it requires our kernel to be shift-invariant (sometimes also referred to as stationary) as stated in Definition 2.

Definition 2 (Shift-Invariant)**.** *A kernel $k : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{C}$ is called shift-invariant if $k(\boldsymbol{x}, \boldsymbol{y}) = g(\boldsymbol{x} - \boldsymbol{y})$ for some positive definite function $g : \mathbb{R}^N \to \mathbb{C}$* [HAe16, page 3].

Clearly, the Gaussian RBF kernel is shift-invariant since it only relies on the bounding radius of $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, from Bochner's theorem, a positive definite shift-invariant kernel with $k(0) = 1$ can be computed

as

$$k(\boldsymbol{x}-\boldsymbol{y}) = \int_{\mathbb{C}^d} \exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right) p(\boldsymbol{\omega})\, d\boldsymbol{\omega}. \tag{2}$$

The main idea of RFF is to approximate the integral in 2 using the following Monte-Carlo estimate

$$\begin{aligned} k(\boldsymbol{x}-\boldsymbol{y}) &= \int_{\mathbb{C}^d} \exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right) p(\boldsymbol{\omega})\, d\boldsymbol{\omega} \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim p(\cdot)}\left(\exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right)\right) \\ &\simeq \frac{1}{D}\sum_{j=1}^{D} \exp\left(i\langle \boldsymbol{\omega}_j, \boldsymbol{x}-\boldsymbol{y}\rangle\right) \\ &= \sum_{j=1}^{D} \left(\frac{1}{\sqrt{D}}\exp\left(i\langle \boldsymbol{\omega}_j, \boldsymbol{x}\rangle\right)\right)\overline{\left(\frac{1}{\sqrt{D}}\exp\left(i\langle \boldsymbol{\omega}_j, \boldsymbol{y}\rangle\right)\right)} \\ &= \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y})\rangle_{\mathbb{C}^D} \end{aligned}$$

where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$ using the feature map

$$\varphi(\boldsymbol{x}) = \frac{1}{\sqrt{D}}\left[z(\boldsymbol{\omega}_1, \boldsymbol{x}), z(\boldsymbol{\omega}_2, \boldsymbol{x}), \ldots, z(\boldsymbol{\omega}_D, \boldsymbol{x})\right]^{\intercal} \tag{3}$$

where for convenience of notation $z(\boldsymbol{\omega}, \boldsymbol{x}) = \exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right)$. This allows the Gram matrix to be estimated as $\boldsymbol{K} \simeq \widetilde{\boldsymbol{K}} = \boldsymbol{Z}\boldsymbol{Z}^{\intercal}$ where $\boldsymbol{Z} = [\varphi(\boldsymbol{x}_1), \varphi(\boldsymbol{x}_2), \ldots \varphi(\boldsymbol{x}_D)] \in \mathbb{C}^{n\times D}$ [Rah08, Liu21, HAe16]. To simplify computation in most settings both $p(\cdot)$ and $k(\Delta)$ are real valued functions meaning $\exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right)$ can replaced with its real component $\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right)$. The vast majority of literature uses the embeddings Rahimi and Recht provide for $\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right)$ where $z(\boldsymbol{\omega}, \boldsymbol{x})$ satisfies equation 2. The first embedding takes the form

$$z(\boldsymbol{\omega}, \boldsymbol{x}) = \left[\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right), \sin\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right)\right]^{\intercal} \tag{4}$$

which satisfies 2 since

$$\begin{aligned} & z(\boldsymbol{\omega}, \boldsymbol{x})^{\intercal} z(\boldsymbol{\omega}, \boldsymbol{y}) \\ &= \left[\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right), \sin\left(\langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right)\right] \begin{bmatrix} \cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right) \\ \sin\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right) \end{bmatrix} \\ &= \cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right)\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right) + \sin\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right)\sin\left(\langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right) \\ &= \frac{1}{2}\left(\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle + \langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right) + \cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle - \langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right)\right) + \\ &\qquad \frac{1}{2}\left(\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle - \langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right) - \cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle + \langle \boldsymbol{\omega}, \boldsymbol{y}\rangle\right)\right) \\ &= \cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right). \end{aligned}$$

The other embedding Rahimi and Recht give is

$$z(\boldsymbol{\omega}, \boldsymbol{x}) = \sqrt{2}\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle + b\right) \tag{5}$$

where $b \sim U[0, 2\pi]$. Using a similar argument we can show that this embedding also satisfies 2. However, Sutherland and Schneider [DJSaJS15] argue that the Gaussian RBF kernel is better suited for the

embedding given in 4. To summarise their argument we denote

$$\varphi_1(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x} \rangle\right) \\ \cos\left(\langle \boldsymbol{\omega}_2, \boldsymbol{x} \rangle\right) \\ \vdots \\ \cos\left(\langle \boldsymbol{\omega}_{D/2}, \boldsymbol{x} \rangle\right) \\ \sin\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x} \rangle\right) \\ \vdots \\ \sin\left(\langle \boldsymbol{\omega}_{D/2}, \boldsymbol{x} \rangle\right) \end{bmatrix} \tag{6}$$

to be the feature map corresponding to embedding in equation 4 and

$$\varphi_2(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x} \rangle + b_1\right) \\ \vdots \\ \cos\left(\langle \boldsymbol{\omega}_D, \boldsymbol{x} \rangle + b_D\right) \end{bmatrix} \tag{7}$$

to be the feature map corresponding to equation 5. They then show that

$$\begin{aligned} \mathbb{V}\left[\varphi_1(\Delta)\right] &= \frac{1}{D}\left(1 + k(2\Delta) - 2k(\Delta)^2\right) \\ \mathbb{V}\left[\varphi_2(\Delta)\right] &= \frac{1}{D}\left(1 + \frac{1}{2}k(2\Delta) - k(\Delta)^2\right) \end{aligned}$$

meaning the variance of φ_1 is smaller whenever

$$\mathbb{V}\left[\cos\left(\langle \boldsymbol{\omega}, \Delta \rangle\right)\right] = \frac{1}{2} + \frac{1}{2}k(2\Delta) - k(\Delta)^2 \leq \frac{1}{2}.$$

When using the Gaussian kernel,

$$\mathbb{V}\left[\cos\left(\langle \boldsymbol{\omega}, \Delta \rangle\right)\right] = \frac{1}{2}\left(1 - \exp\left(-\frac{2\left\|\Delta\right\|_2^2}{\sigma^2}\right)\right)^2 \leq \frac{1}{2}$$

so that $\varphi_1(\Delta) \leq \varphi_2(\Delta)$ for any $\Delta \in \mathbb{R}^d$. There finding were indeed consistent with our preliminary results. With this in mind, an embedding of φ_1 was always used for our experiments.

Another important result Rahimi and Recht show provides a bound on the sup-norm of the difference between a Gram matrix and its RFF approximation stated in Proposition 3.

Proposition 3. *Let $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y}) = k(\Delta)$ be a continuous shift-invariant, positive definite function defined on compact subset $\mathcal{M} \subset \mathbb{R}^d$ having radius ℓ where $k(0) = 1$ such that $\nabla^2 k(0)$ exists. Then for the feature mapping defined in equation 6 let $\sigma_p^2 = \mathbb{E}_{\boldsymbol{\omega} \sim p(\cdot)} \left\|\boldsymbol{\omega}\right\|_2^2 = \operatorname{tr} \nabla^2 k(0)$ then for any $\varepsilon \in \mathbb{R}_{>0},\ \varepsilon \leq \sigma_p \ell$ we have*

$$\mathbb{P}\left[\sup_{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{M}} \left|\langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle_{\mathbb{R}^D} - k(\boldsymbol{x}, \boldsymbol{y})\right| \geq \varepsilon\right] \leq \alpha\left(\frac{\sigma_p \ell}{\varepsilon}\right)^2 \exp\left(-\frac{D\varepsilon^2}{8(d+2)}\right)$$

where $\alpha \in \mathbb{R}_{>0},\ \alpha < \infty$ does not depend on anything [Rah08, page 3].

Rahimi and Recht prove Proposition 3 for $\alpha = 2^8$ although Sutherland and Schneider improve this to $\alpha = 66$ [DJSaJS15, page 3]. Observe that this bound is somewhat determined by the ratio D/d which is why D is often chosen as a multiple of d.

These results justify the RFF procedure seen in Algorithm 1, which was used to approximate a Gram matrix for the data set $\boldsymbol{X}$ using the feature map from 6.

Algorithm 1: RFF Algorithm

input : $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, the dimension of the feature space D.
output: $\widetilde{\boldsymbol{K}} \simeq \boldsymbol{K}$ where $\boldsymbol{K}$ is the Gram matrix corresponding to $\boldsymbol{X}$.

Construct $\boldsymbol{W} \triangleq [\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D]^\intercal \in \mathbb{R}^{D \times d}$ where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$
$\boldsymbol{Z} = \frac{1}{\sqrt{D}} \left[\cos\left(\boldsymbol{W}\boldsymbol{X}^\intercal\right), \sin\left(\boldsymbol{W}\boldsymbol{X}^\intercal\right)\right]^\intercal$
$\widetilde{\boldsymbol{K}} = \boldsymbol{Z}\boldsymbol{Z}^\intercal$
return $\widetilde{\boldsymbol{K}}$

Algorithm 1 of course assumes an appropriate construction of $\boldsymbol{W}$, commonly called the transformation matrix, and thus has access to a routine which allows one to sample from $p(\cdot)$. When using the RBF Gaussian kernel, the spectral distribution given in equation 1 corresponds to a multivariate Gaussian distribution with mean $\mathbf{0}$ and covariance matrix $\left(\frac{\sigma}{\sqrt{2}}\right)^{-2} \mathbb{1}_{D \times D}$. This means $\boldsymbol{W}$ can simply be constructed as $\boldsymbol{W} = \left(\frac{\sigma}{\sqrt{2}}\right)^{-1} [\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D]^\intercal$ where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} \mathcal{N}\left(\mathbf{0}, \mathbb{1}_{D \times D}\right)$. Tranformations matrices constructed in this manner are given the notation $\boldsymbol{W}_{\text{RFF}}$. It can be shown that if $\boldsymbol{W}_{\text{RFF}}$ is used as the transformation matrix in Algorithm 1 then it produces an unbiased estimate, $\widetilde{\boldsymbol{K}}_{\text{RFF}}$, for the Gram matrix. This stated more precisely in Lemma 4.

Lemma 4. *$\widetilde{\boldsymbol{K}}_{RFF}$ is an unbiased estimate of $\boldsymbol{K}$, that is*

$$\mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{RFF}\right)_{ij}\right] = \exp\left(\frac{-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2}{2\sigma^2}\right)$$

[Yu16, page 3].

Proof. We shall prove this for the Gaussian kernel, although proofs for other kernels are analogous. Let $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y}$. Recall the kernel is approximated as

$$\sum_{j=1}^{D} \frac{1}{D} \cos\left(\langle \boldsymbol{\omega}_j, \boldsymbol{z} \rangle\right)$$

where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$. By Bochner's theorem

$$\mathbb{E}\left[\cos\left(\langle \boldsymbol{\omega}_j, \boldsymbol{z} \rangle\right)\right] = \exp\left(-\left\|\boldsymbol{z}\right\|_2^2 / 2\sigma\right)$$

meaning $\widetilde{\boldsymbol{K}}_{\text{RFF}}$ provides an unbiased estimate of $\boldsymbol{K}$. □

Unfortunately, constructing the transformation matrix using $\boldsymbol{W}_{\text{RFF}}$ does not scale well as the dimension of the feature space increases. Thus the focus of the upcoming sections will be to highlight a few of the more popular alternative methods used in the literature for the construction of the transformation matrix.

1.2. **Orthogonal Random Features.** In the previous chapter Algorithm 1 assumed some sort of mechanism for producing the transformation matrix $\boldsymbol{W}$. The construction presented in 1.1 involved sampling $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$. For the Gaussian RBF kernel this meant sampling from the multivariate Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbb{1}_{D\times D})$. The transformation matrix constructed in this manner was denoted $\boldsymbol{W}_{\text{RFF}}$. Recently, there has been a buzz in the literature exploring alternative constructs for the transformation matrix described in Section 1.1 [Liu21]. We shall consider the two methods proposed by Yu *et al.* [Yu16]; the first method here and the second in the following section (1.3). The first method from Yu *et al.* is the Orthogonal Random Features (ORF) method with imposes orthogonality on the transformation matrix. To do this a Gaussian matrix $\boldsymbol{G} \in \mathbb{R}^{D\times d}$ is first produced, much like in $\boldsymbol{W}_{\text{RFF}}$. An orthogonal matrix $\boldsymbol{Q}$ is then created by taking the QR-factorization (see **??**) of $\boldsymbol{G}$. However, the random orthogonal matrix, $\boldsymbol{Q}$, will not give an unbiased estimate of the kernel matrix. To fix this, the following common probabilistic identity is employed

$$\|\boldsymbol{z}\|_2^2 \sim \chi_k^2, \text{ where } \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbb{1}_{k\times k})$$

where χ_k^2 is the chi-squared distribution with k degrees of freedom [Bro91, page 41]. This identity is easily demonstrated by equating a shared moment generating function of $(1-2t)^{-\frac{k}{2}}$ for $t < \frac{1}{2}$. Taking the square root of both sides gives $\|\boldsymbol{z}\|_2 \sim \chi_k$ where χ_k is the chi distribution with k degrees of freedom. In the RFF method, each $\boldsymbol{\omega}_i \in \mathbb{R}^D$ was independently taken from the multivariate normal Gaussian distribution meaning that using the identity provided above $\|\boldsymbol{\omega}_i\|_2 \sim \chi_D$. The ORF method augments $\boldsymbol{Q}$ by scaling its rows by iid χ_D values which can be accomplished through right multiplication with $\boldsymbol{S} = \text{diag}(\psi_1, \psi_2, \ldots, \psi_D)$ where $\psi_i \overset{\text{iid}}{\sim} \chi_D$. This means

$$\left\|(\boldsymbol{SQ})_{(i)}\right\|_2 = \left\|\psi_i \boldsymbol{Q}_{(i)}\right\|_2 = \psi_i \sim \chi_D$$

so that the row norms of $\boldsymbol{G}$ and $\boldsymbol{SQ}$ have the same distribution. Thus the transformation matrix for the ORF method is

$$\boldsymbol{W}_{\text{ORF}} = \left(\frac{\sigma}{\sqrt{2}}\right)^{-1} \boldsymbol{SQ}. \tag{8}$$

The main downside the the ORF method is that the QR-factorization brings a computational cost of $\mathcal{O}(Dd)$. Fortunately when using $\boldsymbol{W}_{\text{ORF}}$ as our transformation matrix in Algorithm 1 the approximate Gram matrix $\widetilde{\boldsymbol{K}}_{\text{RFF}}$ is an unbiased estimate of $\boldsymbol{K}$, stated more formally in Theorem 5.

Theorem 5. *$\widetilde{\boldsymbol{K}}_{ORF}$ is an unbiased estimate of $\boldsymbol{K}$, that is*

$$\mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{ORF}\right)_{ij}\right] = \exp\left(\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{\sigma^2}\right)$$

[Yu16, page 3].

Furthermore, the variance of $\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}$ is bounded by

$$\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}\right] - \mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right] = \frac{1}{D}\left(\frac{g(\tau)}{d} - \frac{(d-1)e^{-\tau^2}\tau^4}{2d}\right)$$

where $\tau = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 / \frac{\sigma}{\sqrt{2}}$ and

$$g(\tau) = \frac{e^{\tau^2}\left(\tau^8 + 6\tau^6 + 7\tau^4 + \tau\right)}{4} + \frac{e^{\tau^2}\tau^4\left(\tau^6 + 2\tau^4\right)}{2d}$$

[Liu21, page 8]. This shows that there are scenarios for which $\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}\right] < \mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right]$, namely when d is large and τ is small. Also, the ratio in variance between $\widetilde{\boldsymbol{K}}_{\text{ORF}}$ and $\widetilde{\boldsymbol{K}}_{\text{RFF}}$ for large d can be approximated as

$$\frac{\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}\right]}{\mathbb{V}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right]} \simeq 1 - \frac{(s-1)e^{-\tau^2}\tau^4}{d(1-e^{-\tau^2})^2}$$

[Liu21, page 8].

1.3. **Random Ortho-Matrices and Structured Orthogonal Random Matrices.** The second method we shall consider for producing a transformation matrix also originates from Yu's *et al.* paper, which Choromanski *et al.* [Cho17] generalized as Random Ortho-Matrices (ROM). This second class of methods is underpinned by transformation matrices with the same variance reductions as ORF with the added benefit of time and memory savings. The transformation matrices generated using ROM take the form

$$\boldsymbol{W}_{\text{ROM}} = \sqrt{d}\prod_{i=1}^{k}\boldsymbol{S}\boldsymbol{D}_i \tag{9}$$

where $\boldsymbol{S} \in \mathbb{R}^{D\times D}$ has orthogonal rows and $\boldsymbol{D} = \text{diag}\left(\delta_1, \ldots, \delta_D\right) \in \mathbb{R}^{D\times D}$ where $\delta_i \overset{\text{iid}}{\sim} U\left(\{-1, 1\}\right)$. This matrix can be forced into a $\mathbb{R}^{D\times d}$ sized matrix by simply extracting the first d columns of $\boldsymbol{D}_1$. The matrix to take the role of $\boldsymbol{S}$ in virtually every application of ROM is the Hadamard matrix, defined in 6, which facilitates a fast $m\log(n)$ matrix multiplication with a size $m \times n$ and is known as Fast Walsh-Hadamard transform (FWHT) [FaA76].

Definition 6 (Hadamard Matrix)**.** *The Hadamard matrix $\boldsymbol{H}_i \in \mathbb{R}^{\left(2^{i-1}\times 2^{i-1}\right)}$ is defined recursively as*

$$\boldsymbol{H}_i = \begin{cases} [1] & , i = 1 \\ \frac{1}{\sqrt{2}}\begin{bmatrix} \boldsymbol{H}_{i-1} & \boldsymbol{H}_{i-1} \\ \boldsymbol{H}_{i-1} & -\boldsymbol{H}_{i-1} \end{bmatrix} & , i > 1 \end{cases}.$$

Note that while Hadamard matrices are only defined for dimensions of exact powers of 2, although other sizes can be constructed by removing portions of the matrix given in Definition 6 or by padding with 0. This provides a concrete means for which one can generate a transformation matrix

$$\sqrt{d}\prod_{i=1}^{k}\boldsymbol{H}\boldsymbol{D}_i \tag{10}$$

where $\boldsymbol{H}$ is an appropriately sized Hadamard matrix. It is easy to check that the matrix generated by equation 10 shares the same expected rows norm lengths as $\boldsymbol{W}_{\text{ORF}}$ and thus enjoys the same variance reduction benefits. Moreover, since matrix multiplication with $\boldsymbol{H}$ can be performed in $\mathcal{O}\left(D\log(d)\right)$ time (using FWHT) and multiplication with $\boldsymbol{H}$ can be performed in $\mathcal{O}(D)$ time, the ROM method has the added benefit of improved run time complexity $\mathcal{O}\left(D\log(d)\right)$ using only $\mathcal{O}(D)$ extra memory. Table 2 gives a comparison of the time and space complexities for the methods mentioned so far.

Despite the wide use of the ROM method in various machine learning tasks [Cho17, And15, Cho20, Liu21] a number of high-interest theoretical properties remain unsolved, leaving many aspects of this method shrouded in mystery. Instead, much of what we understand about ROM's estimate capabilities

TABLE 2. A comparison of various methods for computing a suitable transformation matrix with the Random Fourier Features paradigm. Typically the dimension of the feature space, D, is chosen as some multiple of the dimension of data, d.

Method	*Time*	*Extra Space*
RFF [Rah08]	$\mathcal{O}(Dd)$	$\mathcal{O}(Dd)$
ORF [Yu16]	$\mathcal{O}(Dd)$	$\mathcal{O}(Dd)$
ROM (SORF) [Cho17, Yu16]	$\mathcal{O}\left(D\log(d)\right)$	$\mathcal{O}(D)$

comes from empirical analysis. Nonetheless, we shall still cover a smaller number of important results that have been established.

Choromanski *et al.* [Cho17] show that there are diminishing returns (estimate wise) for choosing larger values of k in equation 10. They also show that choosing odd values of k in 10 provides better estimates then its even-parity $k-1$ and $k+1$ counterparts. For this reason a k value of 3 is usually chosen which gives rise to the transformation matrix estimate given in equation 11. The method for constructing transformation matrices in this manner is referred to as Structured Orthogonal Random Features (SORF).

$$\boldsymbol{W}_{\text{SORF}} = \sqrt{d}\boldsymbol{H}\boldsymbol{D}_3\boldsymbol{H}\boldsymbol{D}_2\boldsymbol{H}\boldsymbol{D}_1 \tag{11}$$

This is the same transformation matrix estimate that Yu *et al.* provides. Unfortunately using the SORF method in Algorithm 1 does not produce an unbiased estimate of the Gram matrix; however, it does satisfy an asymptotic unbiased property

$$\left|\mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{\text{SORF}}\right)_{ij}\right] - \mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{\text{RFF}}\right)_{ij}\right]\right| \leq \frac{6\tau}{\sqrt{d}}$$

where τ is again $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 / \frac{\sigma}{\sqrt{2}}$ [Liu21, page 8].

Bojarski *et al.* [Boj16, page 4] give an intuitive explanation for the roles of each of the different blocks $\boldsymbol{H}\boldsymbol{D}_1$, $\boldsymbol{H}\boldsymbol{D}_2$ and $\boldsymbol{H}\boldsymbol{D}_3$. The first block can be shown to satisfy

$$\mathbb{P}\left[\|\boldsymbol{H}\boldsymbol{D}_1\boldsymbol{x}\|_\infty > \frac{\log D}{\sqrt{D}}\right] \leq 2d\exp\left(-\frac{\log^2 D}{8}\right), \quad \boldsymbol{x} \in \mathbb{R}^D$$

[Liu21, page 8] so that it can be thought as a "balancer" leaving no single dimension bearing too much of the l^2 norm. For the second block, the cost of using a structured matrix is the loss of independence. The purpose of the second block is to mitigate this effect by making similar input vectors near-orthogonal. Finally the third block controls the capacity of the entire structure by providing a vector of parameters. Near-independence is now implied by the near-orthogonality (achieved by $\boldsymbol{H}\boldsymbol{D}_2$) and the fact that the projections of the Gaussian vector or Radamacher vector onto "almost orthogonal directions" are "close to independent". These roles are portrayed visually in Figure 2.

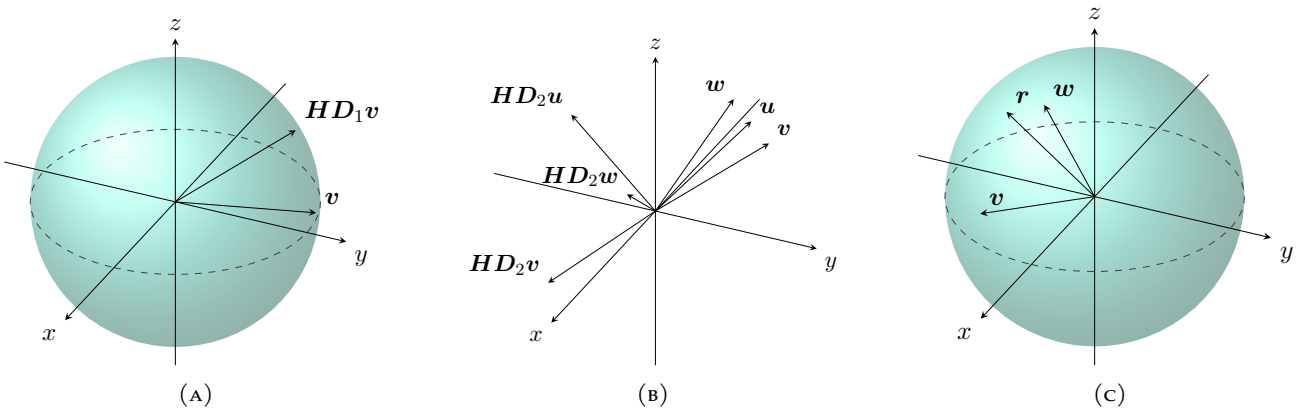


FIGURE 2. A visual representation for the roles of each matrix block in the SORF method. The first block $\boldsymbol{HD}_1$ rotates $\boldsymbol{v}$ so that single dimension bears too much of the l^2 norm seen in panel (A). In panel (B) the second block $\boldsymbol{HD}_2$ transforms vectors so that their image is near-orthogonal. Panel (C) shows that the projection of a random vector $\boldsymbol{r}$ onto two near-orthogonal vector $\boldsymbol{v}, \boldsymbol{w}$ yields a near-independent vector.

2. Applications and Results

So far we have discussed a number of different approximation techniques to use to speed up some of the bottle necks within our Gaussian Process regression and classification algorithms. Here, we shall see how they can be used within the naive implementations from **??** to provide faster methods, at the expense of small amounts of accuracy. We shall also go through an extensive treatment of how the various methods reviewed in previous chapters were implemented and the setup of the experiments used to make comparisons to determine the most successful model.

2.1. Gaussian Processes Prediction Reviewed.

In chapter **??**, a naive implementation for Gaussian Process prediction was presented in **??**. While this does provide a simple and convenient way to produce predictions for a regression task, it is not quite all smooth sailing from here. Unfortunately, there are a number of problems with this algorithm in terms of scalability. For convenience, this algorithm has been restated below but this time with its various bottlenecks highlighted in red.

Algorithm 1: Unoptimized GPR

input : Observations $\boldsymbol{X}, \boldsymbol{y}$ and a test input $\boldsymbol{x}_\star$.
output: A prediction $\overline{f_\star}$ with its corresponding variance $\mathbb{V}[f_\star]$.

1 $\boldsymbol{L} = \text{cholesky}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)$
2 $\boldsymbol{\alpha} = \text{lin-solve}\left(\boldsymbol{L}^\intercal, \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{y}\right)\right)$
3 $\overline{f_\star} = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \boldsymbol{\alpha}$
4 $\boldsymbol{v} = \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}}\right)$
5 $\mathbb{V}[f_\star] = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{x}_\star} - \boldsymbol{v}^\intercal \boldsymbol{v}$
6 **return** $\overline{f_\star}, \mathbb{V}[f_\star]$

To start computing the kernel matrix, $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}$, on line 1 carries an $\mathcal{O}\left(n^2\right)$ runtime since $\mathcal{O}\left(n^2\right)$ pairwise kernel evaluations must be made. Moreover, solving linear systems using a cholesky decomposition (seen on lines $1, 2$ and 4) will incur a runtime of $\mathcal{O}\left(n^3\right)$. While this quadratic scaling may not be much of an issue for smaller datasets, this will scale very poorly where most modern desktops could not handle training datasets any larger than 10^5 with samples [WJMaSKaAGW21, page 2]. The Gaussian Process classifier from **??** suffers for the exact same reasons as GPR. To mitigate the computational burden of these bottle necks, we can replace these procedures with their approximative counterparts discussed in Chapters **??**,1 and **??**.

To start, computing the kernel matrix can be done using either the Nystrom method or the RFF technique, both of which provide better asymptotic runtimes. Similarly, the Cholesky decomposition and linear solves can be replaced with either CG or MINRES to, again, improve runtime performance. This especially makes sense approximating the kernel matrix as the Cholesky decomposition will provide an almost-exact solution to solving these linear system, as this is rather counterintuitive if approximations are used earlier on within the algorithm.

2.2. Experimental Setup.

Each method was implemented in python3 and run using the python3.8.5 interpreter distributed by the official python website. Various scientific computing python libraries, such

as numpy, pandas and scipy, were used to help implement methods to enhance runtime performance. A just-in-time compiler was also used to improve the performance of the FWHT and Krylov-Subspace methods as fast runtime is critically important for the success of both these methods. Experiments were carried out on a variety of different datasets listed in Table 3.

TABLE 3. Descriptions and sources for each of the datasets used in experiments.

Name	*Description*	d	n	*Source*
3DSN	The 3D Spatial Network (3DSN) dataset was constructed by adding elevation information to a 2D road network in North Jutland, Denmark (covering a region of $185 \times 135 km^2$).	2	2000	UCI
Abalone	Physical measurements of abalones.	7	4177	UCI
magic04	Simulated registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope.	10	19020	UCI
Wine	Chemical measurements of wine.	11	4898	UCI
Temp	Weather station data from rural Queensland.	1	11324	Qld Gov
Stocks	Daily stock prices spanning 2000 to 2019.	4	4904	tiskw

2.2.1. *Kernel Martix Approximation Testing.* To test the various kernel matrix approximation techniques, each method was used to construct an estimate of the actual kernel matrix with varying sample sizes. The definition of samples changes depending of which family of approximation technique is considered. For the Nystrom technique, the number of samples refers to s, that is the number of columns sampled. For the RFF technique the samples refer to D, that being the dimension of the constructed feature space. The Nystrom technique was implemented according to [PDaMWM05] and RFF according to [Rah08] and [Liu21]. Each method was used to compute approximations for kernel matrices for each of the above data sets for a fixed σ and sample size. This was repeated 30 times for each dataset. For every method the relative Frobenius error $\left\|\boldsymbol{K_{XX}} - \widehat{\boldsymbol{K}}_{\boldsymbol{XX}}\right\|_F / \left\|\boldsymbol{K_{XX}}\right\|_F$ and relative infinity error $\left\|\boldsymbol{K_{XX}} - \widehat{\boldsymbol{K}}_{\boldsymbol{XX}}\right\|_\infty / \left\|\boldsymbol{K_{XX}}\right\|_\infty = \left\|\boldsymbol{K_{XX}} - \widehat{\boldsymbol{K}}_{\boldsymbol{XX}}\right\|_\infty$ (in the case of the RBF kernel) was recorded, using the notation that $\widehat{\boldsymbol{K}}_{\boldsymbol{XX}}$ represents an approximated kernel matrix. The values of σ used were 0.1, 1.0 and 10.0, although a σ value of 2.1 was also used for the Wine data set to compare with [PDaMWM05] results. Additionally, computation time and memory usage for each kernel approximation was also recorded. It should be noted, however, that the time and memory spent on constructing probabilites for the Nystrom methods are *not* included in the kernel construction time and have instead been recorded separately in TODO. This is because the probabilites are typically known prior to kernel matrix construction and are simply used to provide an approximation to the kernel matrix instead on needing to save the entire data kernel matrix for new prediction. Thus they act as an efficient means to quickly save and rebuild the data kernel matrix. The errors, time and memory were average across the repetitions for each experiment.

To understand how well each kernel matrix approximation method performs in terms of providing predictions, each method was used to train a Gaussian process with $4/5^{ths}$ of each data set and required to predict the remaining $1/5^{th}$. Predictions for each data set was repeated 15 times across various sample sizes. The Mean Square Error (MSE) and Classification Error was captured for regression and classification tasks respectively. To make comparisons between different approximation methods as fair as possible, the matrix $\left(\widehat{\boldsymbol{K}}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)^{-1}$ was directly inverted to eliminate any error an in-exact linear system solver might introduce. To do this efficiently, since both the Nystrom and RFF methods produce decompositions to their approximations of the form $\widehat{\boldsymbol{K}}_{\boldsymbol{XX}} = \boldsymbol{U}\boldsymbol{W}^{\dagger}\boldsymbol{U}^{\intercal}$ we can make use of the matrix inversion lemma (see **??**) to compute

$$\begin{aligned}\left(\widehat{\boldsymbol{K}}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)^{-1} &= \left(\boldsymbol{U}\boldsymbol{W}\boldsymbol{U}^{\intercal} + \sigma_n^2 \mathbb{1}_{n\times n}\right)^{-1} \\ &= \sigma_n^{-2}\mathbb{1}_{n\times n} + \sigma_n^{-2}\mathbb{1}_{n\times n}\boldsymbol{U}\left(\boldsymbol{W}^{\dagger} + \boldsymbol{U}^{\intercal}\sigma_n^{-2}\mathbb{1}_{n\times n}\boldsymbol{U}\right)^{-1}\boldsymbol{U}^{\intercal}\sigma_n^{-2}\mathbb{1}_{n\times n}.\end{aligned}$$

Only a single value of σ was used to create predictions for each dataset. The value of σ chosen corresponds to the best value of σ out of $0.1, 1.0$ and 10.0 (as well as 2.1 for the Wine dataset) when an exact GP algorithm was used to provide predictions. The time and memory of performing the matrix inverse using the above was recorded. Again, errors, time and memory were average across the repetitions for each different sample.

2.2.2. *Krylov Subspace Methods Approximation Testing.* Similar to the kernel matrix approximation setup, to see how well CG and MINRES compare in providing predictions each method was used to solve the linear system

$$\left(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)\boldsymbol{\alpha} = \boldsymbol{y}$$

within the GPR algorithm and

$$\left(\boldsymbol{K}_{\boldsymbol{XX}} + \boldsymbol{W}^{-1}\right)\boldsymbol{\alpha} = \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}}^{\intercal}$$

within the GPC algorithm in terms of $\boldsymbol{\alpha}$. As before, predictions for each data set was repeated 15 times across various samples. The Mean Square Error (MSE) and Classification Error was captured for regression and classification tasks respectively. To allow for a fair comparison between the different linear solvers, the kernel matrix was constructed in an exact manner to ensure no error was introduced through any other part of the prediction process. As with the kernel matrix approximation prediction set up, only a single value of σ was used to create predictions for each dataset, that being the value of σ that provided the best predictions results using exacts methods. Each method was used to train a Gaussian process with $4/5^{ths}$ of each data set and required to predict the remaining $1/5^{th}$ over all the different datasets and was repeated 15 times for each dataset with a varying number of maximum iterations. The usual metrics such as prediction error, time and memory usuage were recorded for every experiment and averaged across repetitions for each dataset and method.

2.3. **Discussion.**

2.3.1. *Kernel Matrix approximation.* Starting with the performance of the Nystrom methods, assessing Figures 21 to 32, overall the rls method is certainly the best methods among the sampling distributions as it virtually always is among the top three methods for any combination of dataset, k or σ. Interestingly, non-uniform sampling techniques performed better with datasets for which the spectrum of the corresponding kernel matrix was also non-uniform. For example the spectrum for the kernel matrices of the magic04 and Stocks dataset (see Figures 5 and 6) are relatively uniform and from Figures 25 and 27 we find that the non-uniform sampling techniques give rather poor approximations and generally behave just as bad, or even worse, than the naive uniform sampling distribution. In contrast, non-uniform methods really shone for datasets such as the 3DSN and Wine datasets where some of the better methods could provide almost exact kernel matrix approximations (refer to Figures 21, 22, 31 and 32) after a few thousand samples. This phenomena is likely due to the fact that when the spectrum of a matrix is less uniform, it decays faster, meaning that the matrix can be better expressed as a low-rank approximation. Evidently, the majority of non-uniform sampling distributions are intelligent enough to select columns that are best representative in this low rank approximation, most of the time. What is especially interesting about the Abalone dataset is, despite the rls sampling distribution being almost uniform (compared to other sampling distributions, see Figures 10 and 16), the rls methods seems to perform significantly better than uniform selection, shown in Figure 23. The small amount of variation in the rls sampling distribution presumably provides it with a large enough advantage over the uniform method and selects columns that better provides information to the lower rank approximation.

Looking at the RFF methods, we find that the ORF and SORF methods do not really live up to their acclaimed theoretical error bounds. While the ORF and SORF methods do occasionally produce better errors over the standard transformation matrix sampling (see Figures 33 to 44), it is hard to argue using either of these methods as neither them offer no obvious advantage in terms of time and memory saving (see Figures TODO). This may come as a surprise for most with the SORF method under deliverance, even with a JIT compiler was used to speed up various parts of the algorithm. This dissonance between the theoretical bounds presented in Table 2 and these "real world" experiments is that the single line of code to produce the i.i.d. Gaussian matrix for the standard transformation matrix using the numpy python library has been incredibly well engineered and optimized to produce matrix samples brilliantly fast and designed to cope with large scaling. When given large enough datasets, the SORF method likely will overtake the naive sampling method. Unfortunately, due to the memory constraints of the computers used for this project, such large data sets could not be experimented with.

To make a fair comparison in kernel matrix approximation between the RFF and Nystrom methods, the time that each method used to build an approximation was graphed along side the corresponding relative error. In this setup, we can think about each method having some sort of time budget and that its task is to produce the best approximation it can within this time limit, regardless of how many samples it requires or how big of a feature space it uses. Comparing the methods in this manner, the Nystrom family is far better at lowering the Frobenius error in the approximations it produces, while the RFF methods provide approximations with smaller infinity errors (see Figures 45 to 56). In other words, the entries of the approximations produced by Nystrom methods are *on average* better than those from RFF methods, while *in the worst case* the entries of the approximations produced by RFF are better than those from Nystrom methods. This makes sense since much of the theory in the Nystrom methods was aimed

towards lowering the Frobenius errors, while the theory behind the RFF methods was aimed towards lowering the infinity errors. One final point worth mentioning is that RFF methods seemed to require much larger amounts of memory to produce their approximations, sometimes double or even triple the amount of memory the Nystrom methods used (refer to Figures TODO).

References

[Ras06] Carl Edward and Williams Rasmussen Christopher K. I, *Gaussian processes for machine learning / Carl Edward Rasmussen, Christopher K.I. Williams.*, Adaptive computation and machine learning, MIT Press, Cambridge, Mass., 2006 (eng).

[HHF73] H. Howard Frisinger, *Aristotle's legacy in meteorology*, Bulletin of the American Meteorological Society **54** (1973), no. 3, 198–204.

[Yul27] G. Udny Yule, *On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers*, Philosophical transactions of the Royal Society of London. Series A, Containing papers of a mathematical or physical character **226** (1927), no. 636-646, 267–298 (eng).

[Box08] George E. P. and Jenkins Box Gwilym M and Reinsel, *Time series analysis : forecasting and control / George E.P. Box, Gwilym M. Jenkins, Gregory C. Reinsel.*, 4th ed., Wiley series in probability and statistics, John Wiley, Hoboken, N.J., 2008 (eng).

[VdW19] Mark Van der Wilk, *Sparse Gaussian process approximations and applications*, University of Cambridge, 2019.

[Cao18] Yanshuai Cao, *Scaling Gaussian Processes*, University of Toronto (Canada), 2018.

[SD22] Matías and Estévez Salinero-Delgado José and Pipia, *Monitoring Cropland Phenology on Google Earth Engine Using Gaussian Process Regression*, Remote Sensing **14** (2022), no. 1, DOI 10.3390/rs14010146.

[Pot13] Andries and Lawson Potgieter Kenton and Huete, *Determining crop acreage estimates for specific winter crops using shape attributes from sequential MODIS imagery*, International Journal of Applied Earth Observation and Geoinformation **23** (2013), DOI 10.1016/j.jag.2012.09.009.

[NdF13] Nando de Freitas, *University of British Columbia CPSC 540, Lecture notes in Machine Learning*, University of British Columbia, 2013.

[Mur12] Kevin P. Murphy, *Machine learning : a probabilistic perspective / Kevin P. Murphy.*, Adaptive computation and machine learning, MIT Press, Cambridge, MA, 2012 (eng).

[Ber96] Z.G. Sheftel Berezansky G.F, *Functional analysis. Volume 1 / Y.M. Berezansky, Z.G. Sheftel, G.F. Us ; translated from the Russian by Peter V. Malyshev.*, 1st ed. 1996., Operator

Theory: Advances and Applications, 85, Basel ; Boston ; Berlin : BirkhaIuser Verlag, Basel ; Boston ; Berlin, 1996 (eng).

[Tre97] Lloyd N. (Lloyd Nicholas) and Bau Trefethen David, *Numerical linear algebra / Lloyd N. Trefethen, David Bau.*, SIAM Society for Industrial and Applied Mathematics, Philadelphia, 1997 (eng).

[Dem97] James W Demmel, *Applied numerical linear algebra / James W. Demmel.*, Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1997 (eng).

[Ste08] Ingo and Christmann Steinwart Andreas, *Support Vector Machines*, 1st ed. 2008., Information Science and Statistics, Springer New York, New York, NY, 2008 (eng).

[Ber03] Alain and Thomas-Agnan Berlinet Christine, *Reproducing Kernel Hilbert Spaces in Probability and Statistics*, Springer, SpringerLink (Online service), Boston, MA, 2003 (eng).

[Ste99] Michael L Stein, *Interpolation of Spatial Data Some Theory for Kriging / by Michael L. Stein.*, 1st ed. 1999., Springer Series in Statistics, Springer New York : Imprint: Springer, New York, NY, 1999 (eng).

[Bos92] Bernhard and Guyon Boser Isabelle and Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the fifth annual workshop on computational learning theory, 1992, pp. 144–152 (eng).

[Cor95] Corinna Cortes, *Support-Vector Networks*, Machine learning **20** (1995), no. 3, 273 (eng).

[Kro14] Dirk P and C.C. Chan Kroese Joshua, *Statistical Modeling and Computation by Dirk P. Kroese, Joshua C.C. Chan.*, 1st ed. 2014., Springer New York : Imprint: Springer, New York, NY, 2014 (eng).

[Fle00] R Fletcher, *Practical Methods of Optimization*, John Wiley and Sons, Incorporated, New York, 2000 (eng).

[Bis06] Christopher M Bishop, *Pattern recognition and machine learning / Christopher M. Bishop.*, Information science and statistics, Springer, New York, 2006 (eng).

[Spi90] David J and Lauritzen Spiegelhalter Steffen L, *Sequential updating of conditional probabilities on directed graphical structures*, Networks **20** (1990), no. 5, 579–605.

[WJMaSKaAGW21] Wesley J. Maddox and Sanyam Kapoor and Andrew Gordon Wilson, *When are Iterative Gaussian Processes Reliably Accurate?*, CoRR **abs/2112.15246** (2021), available at 2112.15246.

[MWM11] Michael W. Mahoney, *Randomized algorithms for matrices and data*, CoRR **abs/1104.5557** (2011).

[Hal11] Nathan and Martinsson Halko Per-Gunnar and Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM review **53** (2011), no. 2, 217–288.

[PGMaJT21] Per-Gunnar Martinsson and Joel Tropp, *Randomized Numerical Linear Algebra: Foundations and Algorithms*, arXiv, 2021.

[FR20] Fred Roosta, *University of Queensland MATH3204, Lecture notes in Numerical Linear Algebra and Optimisation*, University of Queensland, 2020.

[Dri06] Petros and Kannan Drineas Ravi and Mahoney, *Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication*, SIAM Journal on Computing **36** (2006), no. 1, 132-157, DOI 10.1137/S0097539704442684, available at https://doi.org/10.1137/S0097539704442684.

[PDaMWM17] Petros Drineas and Michael W. Mahoney, *Lectures on Randomized Numerical Linear Algebra*, arXiv, 2017.

[PDaMWM05] Petros Drineas and Michael W. Mahoney, *On the Nystrom Method for Approximating a Gram Matrix for Improved Kernel-Based Learning*, Journal of Machine Learning Research **6** (2005), no. 72, 2153-2175.

[AGaMWM13] Alex Gittens and Michael W. Mahoney, *Revisiting the Nystrom Method for Improved Large-Scale Machine Learning*, CoRR **abs/1303.1849** (2013), available at 1303.1849.

[CMaCM17] Cameron Musco and Christopher Musco, *Recursive Sampling for the Nystrom Method*, arXiv, 2017.

[PDe11] Petros Drineas etal., *Fast approximation of matrix coherence and statistical leverage*, CoRR **abs/1109.3843** (2011).

[MBCaCMaCM15] Michael B. Cohen and Cameron Musco and Christopher Musco, *Ridge Leverage Scores for Low-Rank Approximation*, CoRR **abs/1511.07263** (2015).

[Kum09] Sanjiv and Mohri Kumar Mehryar and Talwalkar, *Sampling techniques for the nystrom method*, Artificial intelligence and statistics, 2009, pp. 304–311.

[Hoa78] David C and Welsch Hoaglin Roy E, *The Hat Matrix in Regression and ANOVA*, The American statistician **32** (1978), no. 1, 17–22 (eng).

[Ala15] Ahmed and Mahoney Alaoui Michael W, *Fast Randomized Kernel Ridge Regression with Statistical Guarantees*, Advances in Neural Information Processing Systems, 2015.

[Kar10] Noureddine El Karoui, *THE SPECTRUM OF KERNEL RANDOM MATRICES*, The Annals of statistics **38** (2010), no. 1, 1–50 (eng).

[Pre92] William H. (William Henry) Press, *Numerical recipes in C : the art of scientific computing / William H. Press ...* [*et al.*], 2nd ed., Cambridge University Press, Cambridge, 1992 (eng).

[Wan] Guorong and Wei Wang Yimin and Qiao, *Generalized Inverses: Theory and Computations*, Developments in Mathematics, vol. 53, Springer Singapore, Singapore (eng).

[Gre97] Anne Greenbaum, *Iterative methods for solving linear systems Anne Greenbaum.*, Frontiers in applied mathematics ; 17, Society for Industrial and Applied Mathematics SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104, Philadelphia, Pa., 1997 (eng).

[Cho07] Sou-Cheng (Terrya) Choi, *Iterative methods for singular linear equations and least - squares problems*, ProQuest Dissertations Publishing, 2007 (eng).

[CHO11] Sou-Cheng T and PAIGE CHOI Christopher C and SAUNDERS, *MINRES-QLP: A KRYLOV SUBSPACE METHOD FOR INDEFINITE OR SINGULAR SYMMETRIC SYSTEMS*, SIAM journal on scientific computing **33** (2011), no. 3-4, 1810–1836 (eng).

[Rah08] Ali and Recht Rahimi Benjamin, *Random Features for Large-Scale Kernel Machines*, Advances in Neural Information Processing Systems, 2008.

[Pot21] Andres and Wu Potapczynski Luhuan and Biderman, *Bias-Free Scalable Gaussian Processes via Randomized Truncations* (2021) (eng).

[Hah33] Hans Hahn, *S. Bochner, Vorlesungen über Fouriersche Integrale: Mathematik und ihre Anwendungen, Bd. 12.) Akad. Verlagsges., Leipzig 1932, VIII. u. 229S. Preis brosch. RM 14,40, geb. RM16*, Monatshefte für Mathematik **40** (1933), no. 1, A27–A27 (ger).

[Liu21] Fanghui and Huang Liu Xiaolin and Chen, *Random Features for Kernel Approximation: A Survey on Algorithms, Theory, and Beyond*, IEEE transactions on pattern analysis and machine intelligence **PP** (2021) (eng).

[HAe16] Haim Avron etal, *Quasi-Monte Carlo Feature Maps for Shift-Invariant Kernels*, Journal of Machine Learning Research **17** (2016), no. 120, 1-38.

[DJSaJS15] Danica J. Sutherland and Jeff Schneider, *On the Error of Random Fourier Features*, 2015.

[Yu16] Felix X and Suresh Yu Ananda Theertha and Choromanski, *Orthogonal Random Features* (2016) (eng).

[Bro91] Peter J and Davis Brockwell Richard A, *Time Series: Theory and Methods*, Second Edition., Springer Series in Statistics, Springer New York, SpringerLink (Online service), New York, NY, 1991 (eng).

[Cho17] Krzysztof and Rowland Choromanski Mark and Weller, *The Unreasonable Effectiveness of Structured Random Orthogonal Embeddings* (2017) (eng).

[FaA76] Fino and Algazi, *Unified Matrix Treatment of the Fast Walsh-Hadamard Transform*, IEEE transactions on computers **C-25** (1976), no. 11, 1142–1146 (eng).

[And15] Alexandr and Indyk Andoni Piotr and Laarhoven, *Practical and Optimal LSH for Angular Distance* (2015) (eng).

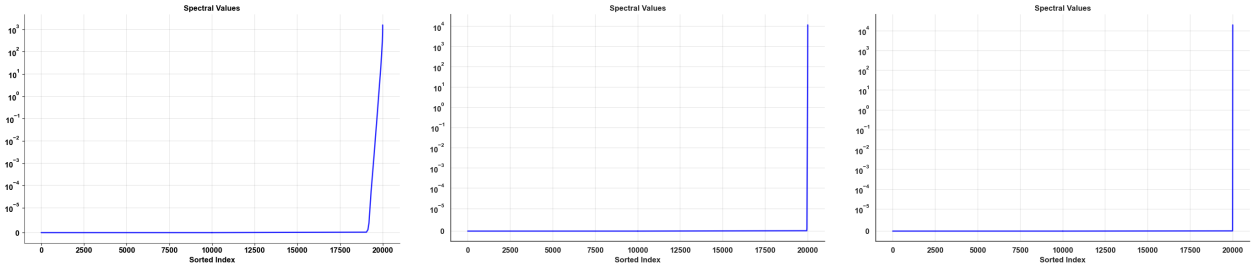
[Cho20] Krzysztof and Likhosherstov Choromanski Valerii and Dohan, *Rethinking Attention with Performers* (2020) (eng).

[Boj16] Mariusz and Choromanska Bojarski Anna and Choromanski, *Structured adaptive and random spinners for fast machine learning computations* (2016) (eng).

Appendix A. Supplementary Results

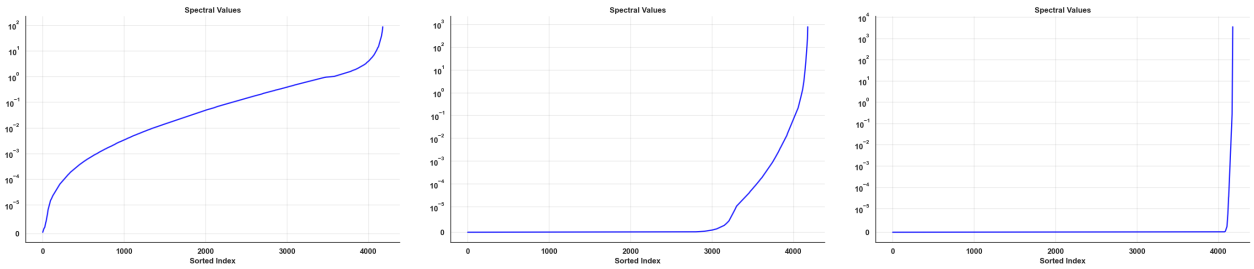
Additional results that may have not been included in the main text for conciseness.

A.1. **Gram Matrix Spectral Values.**



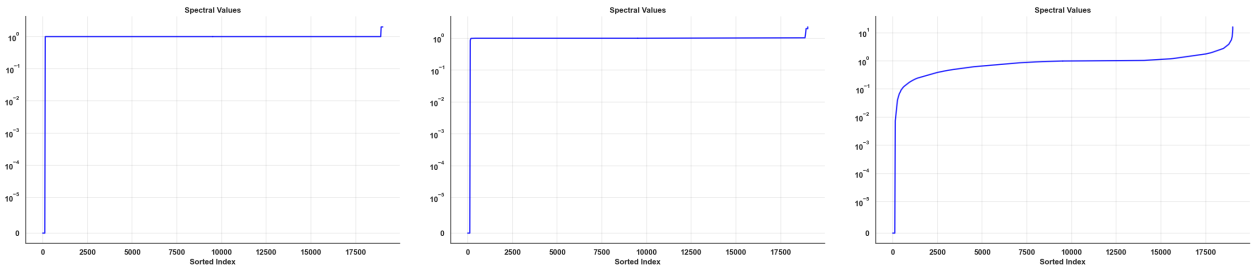
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 3. Spectral values for 3D-spatial network data.



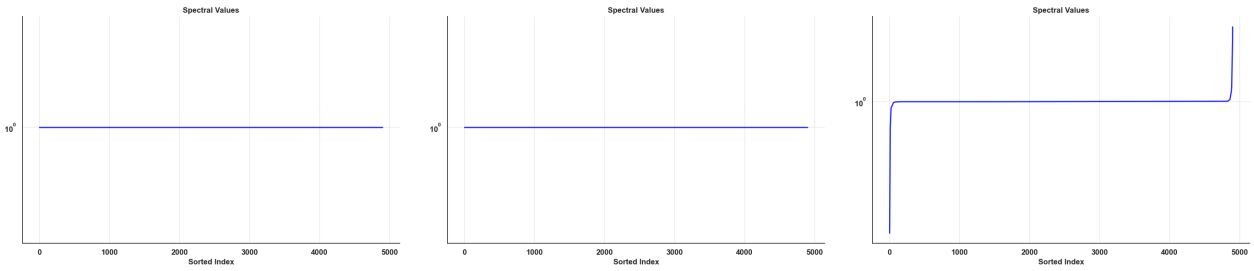
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 4. Spectral values for abalone data.



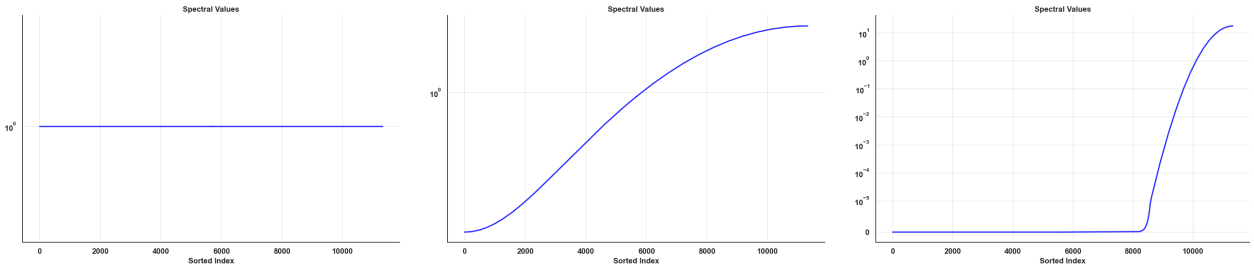
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 5. Spectral values for magic04 data.



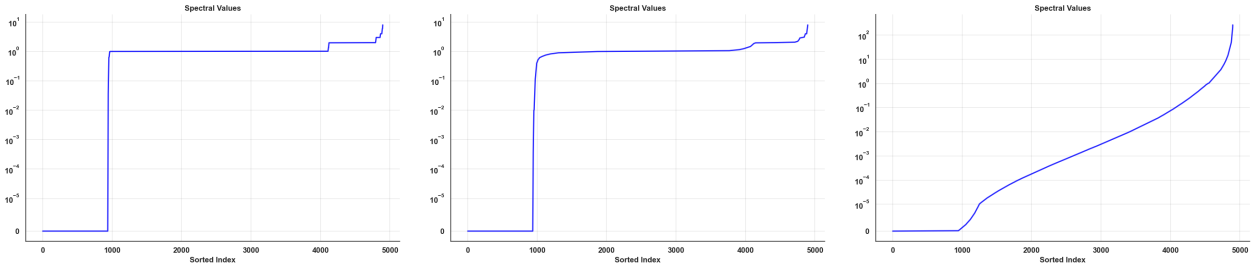
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 6. Spectral values for stock market data.



(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

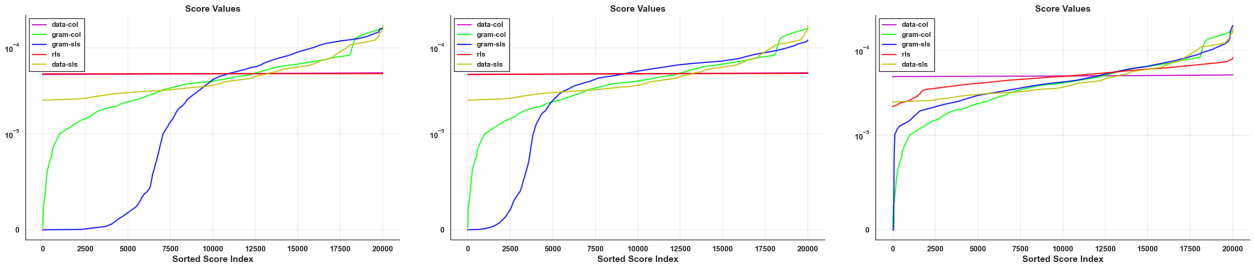
FIGURE 7. Spectral values for temperature data.



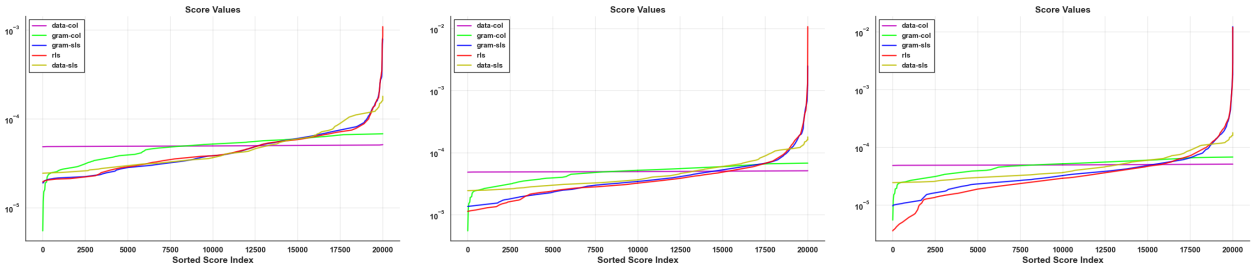
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 8. Spectral values for wine data.

A.2. **Nystrom Scores.**

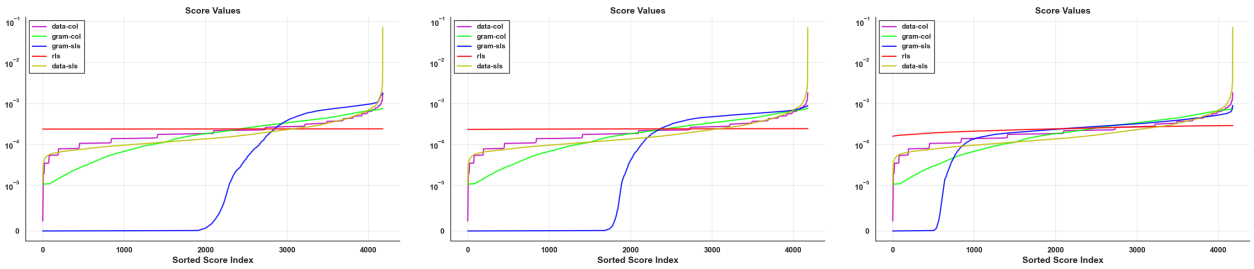


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

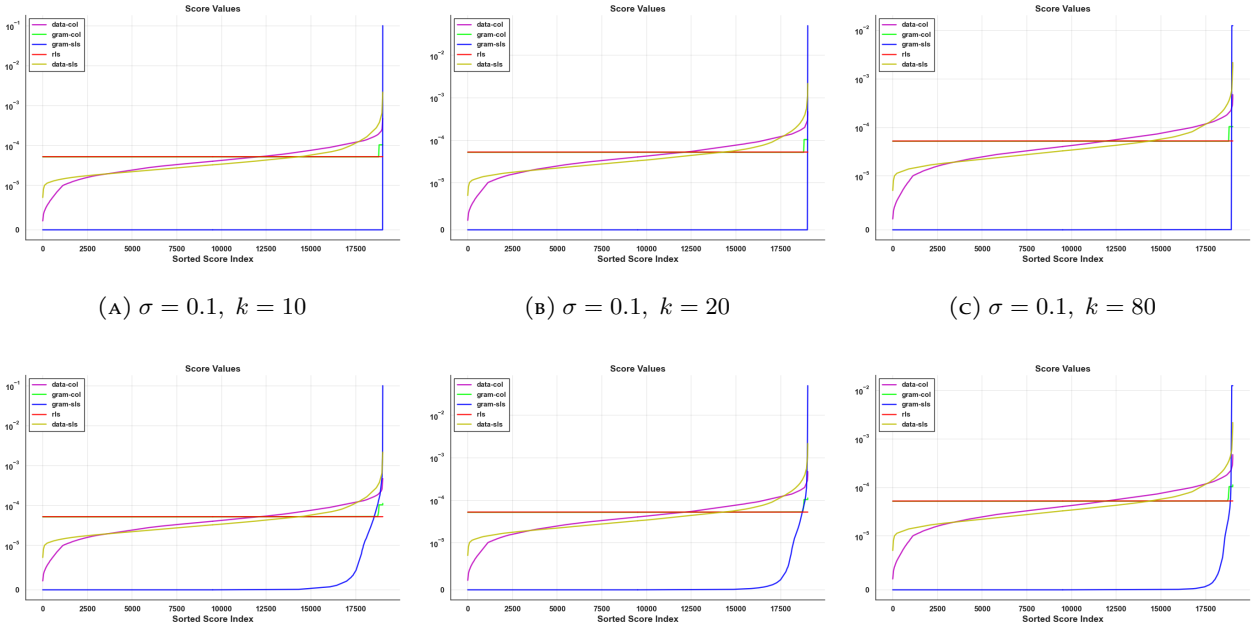
FIGURE 9. Nystrom scores for the 3D-spatial network data set.



(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

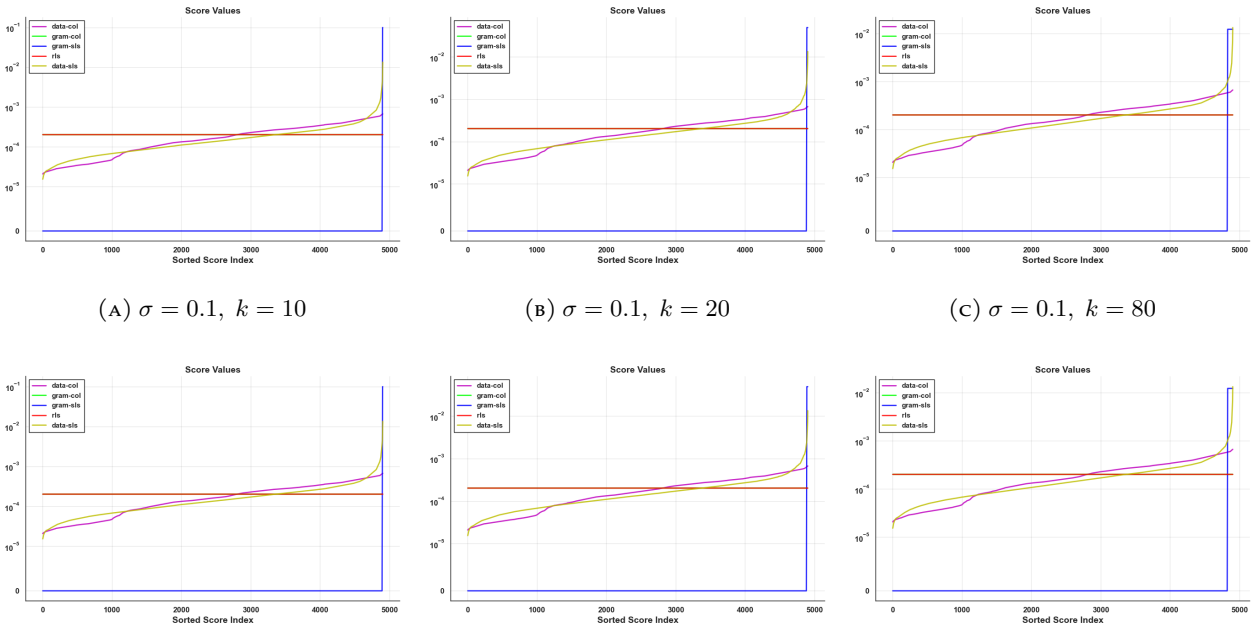
FIGURE 10. Nystrom scores for the Abalone data set.



(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

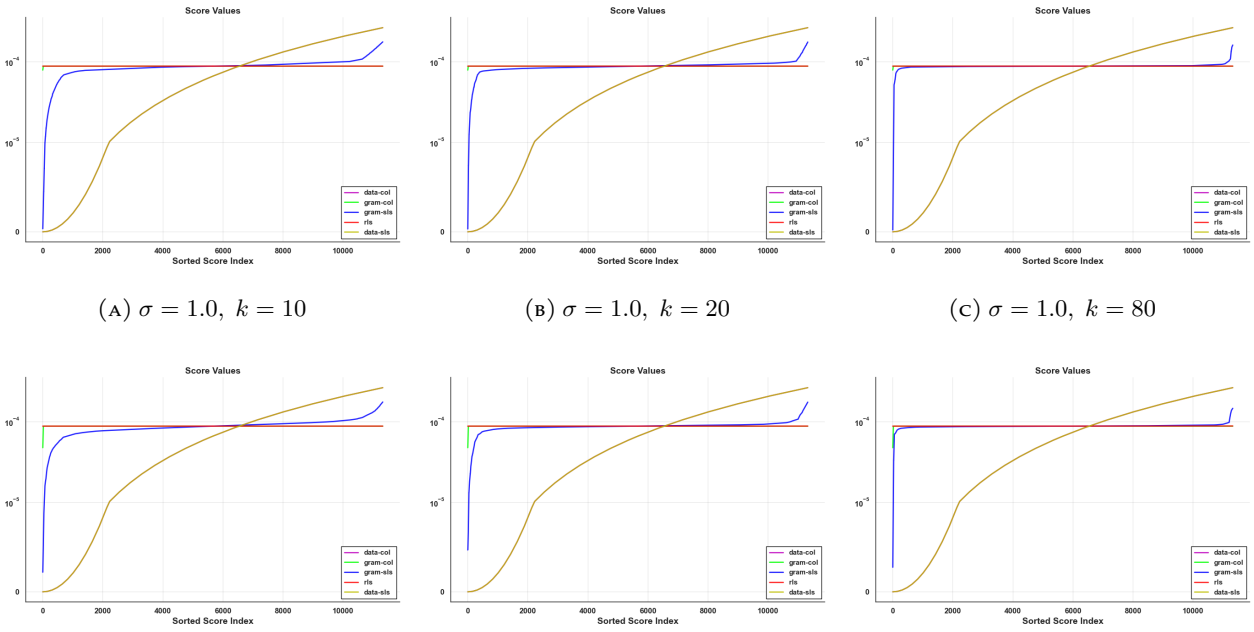
FIGURE 11. Nystrom scores for the Magic data set.



(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

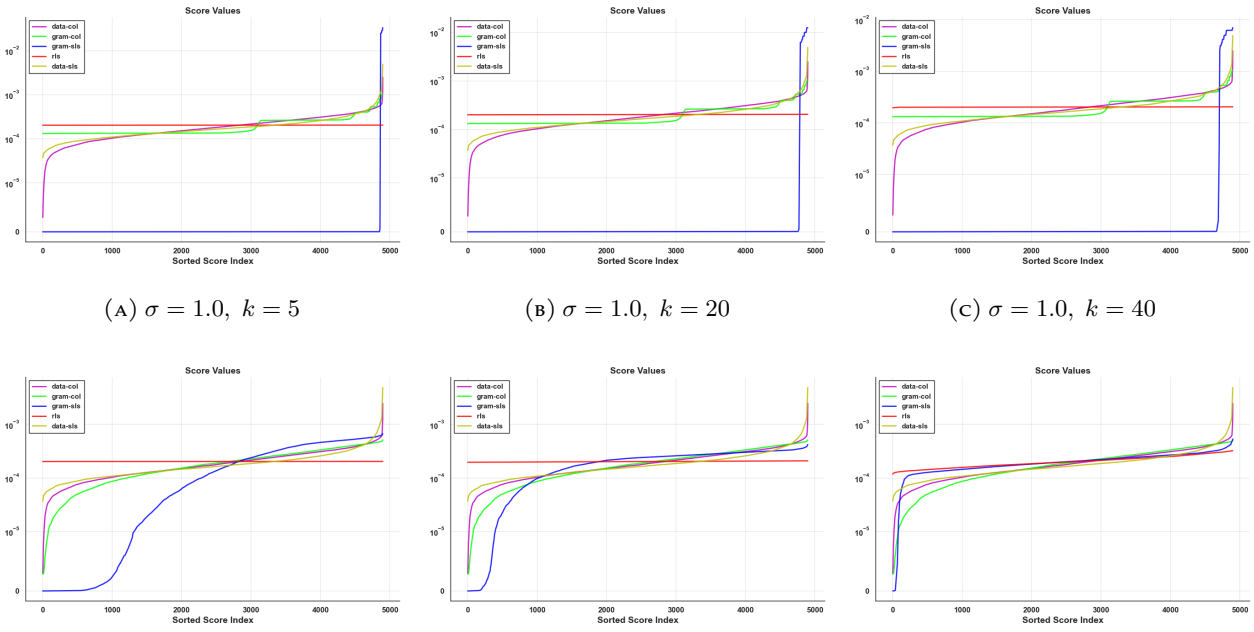
FIGURE 12. Nystrom scores for the stock market data set.



(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 13. Nystrom scores for the temperature data set.

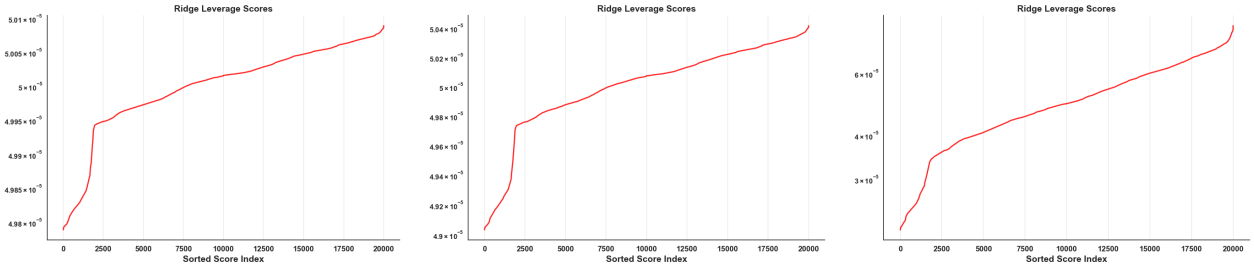


(A) $\sigma = 1.0,\ k = 5$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 40$

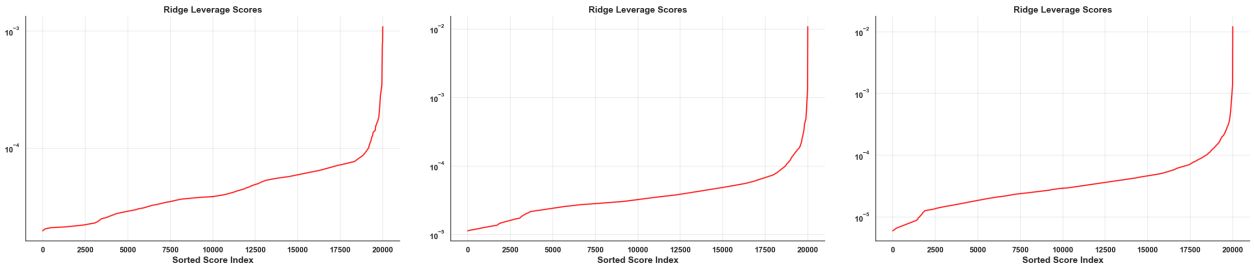
(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 14. Nystrom scores for the wine data set.

A.3. **Ridge Leverage Scores.**

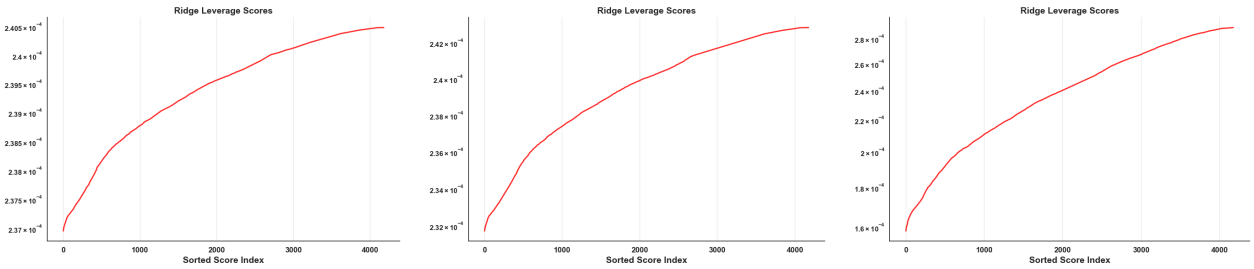


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

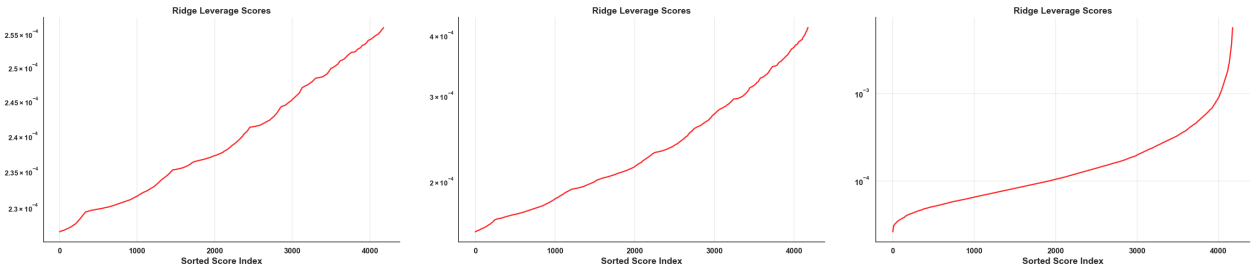


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 15. Ridge Leverage scores for the 3D-spatial network data set.

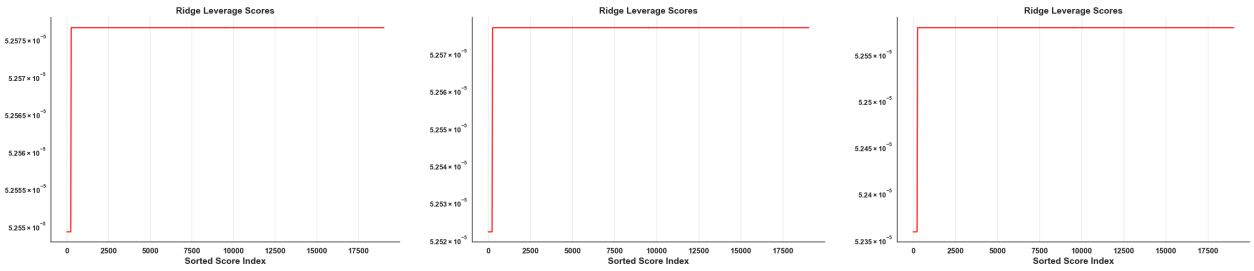


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

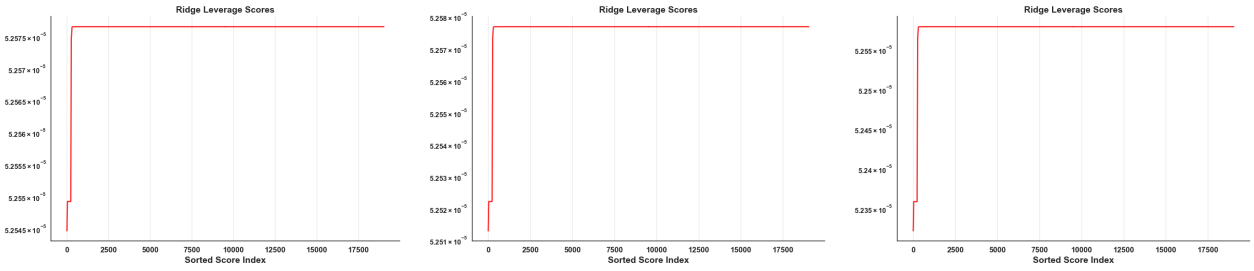


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 16. Ridge Leverage scores for the Abalone data set.

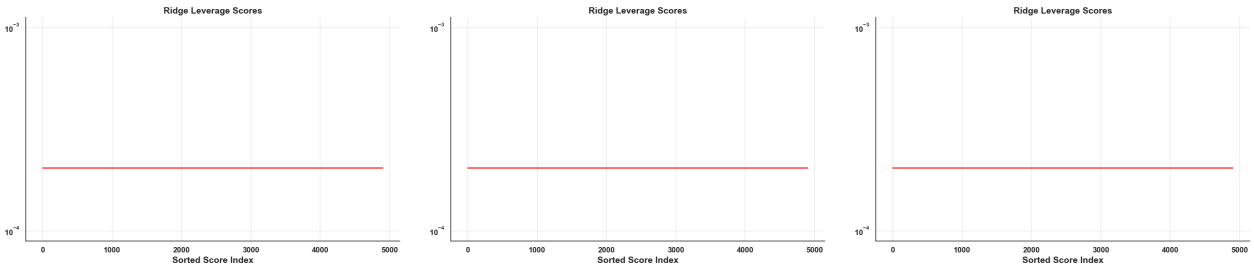


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

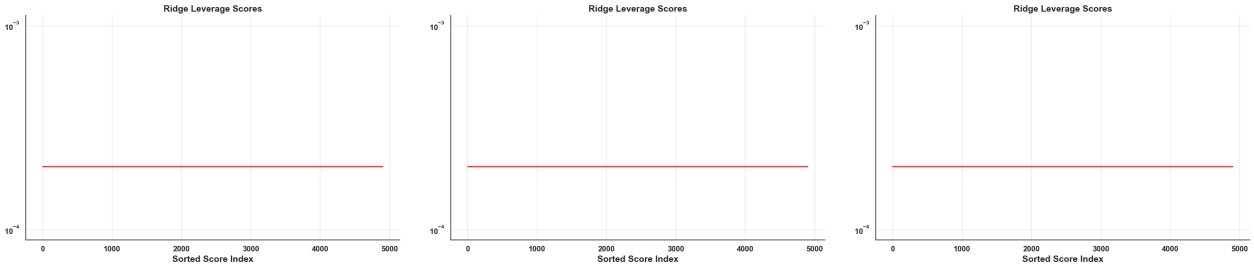


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 17. Ridge Leverage scores for the Magic data set.

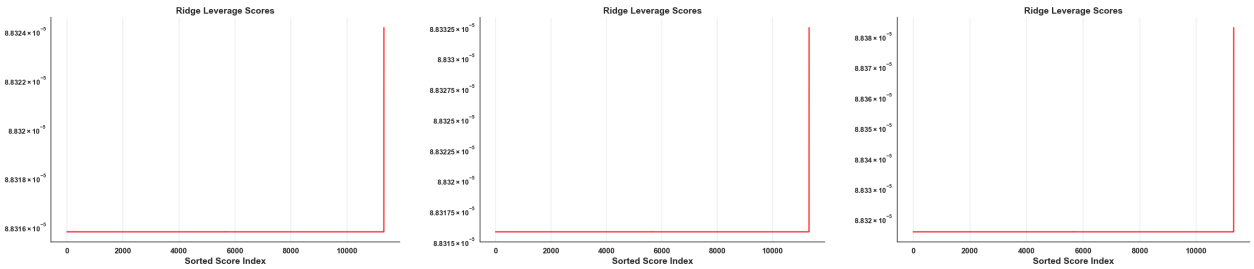


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

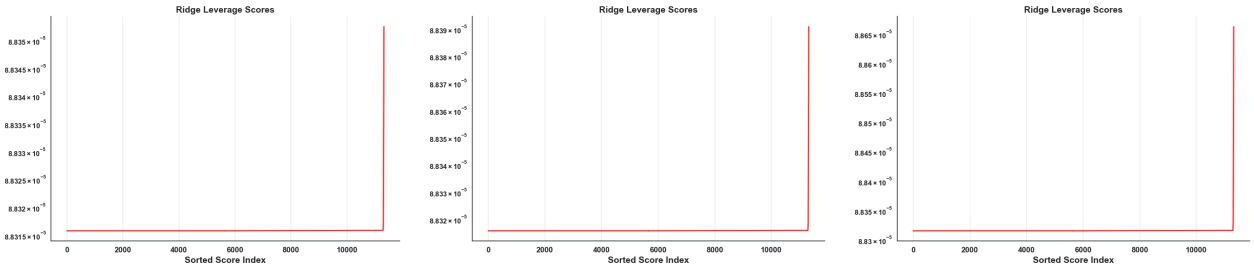


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 18. Ridge Leverage scores for the stock market data set.

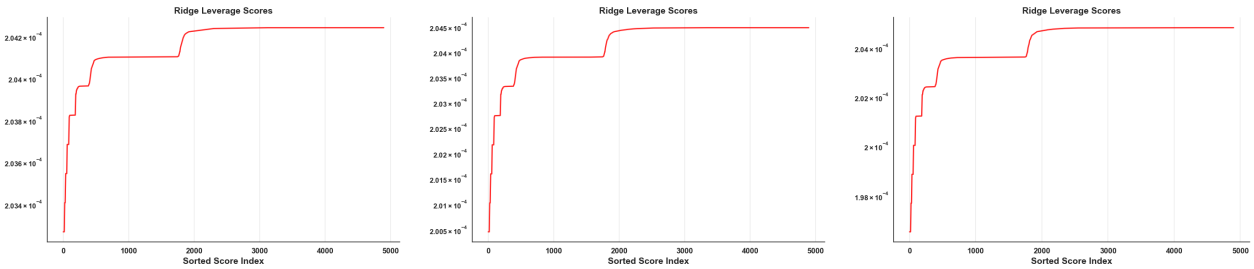


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

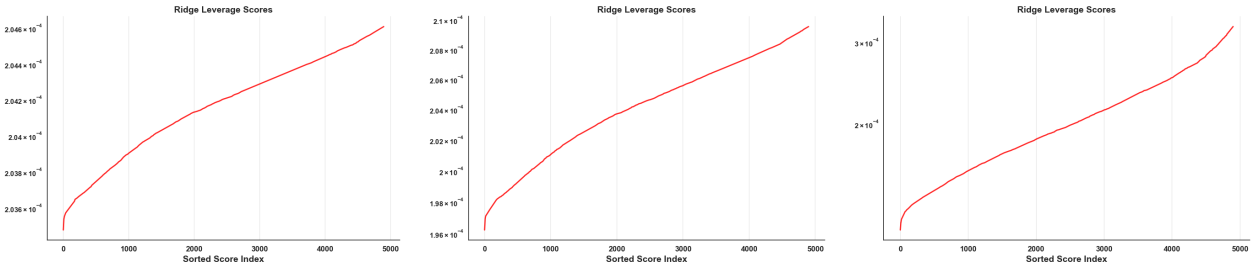


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 19. Ridge Leverage scores for the temperature data set.



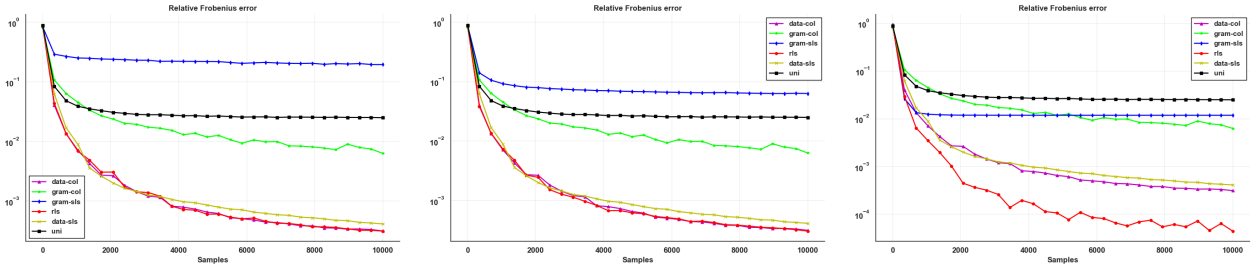
(A) $\sigma = 1.0,\ k = 5$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 40$



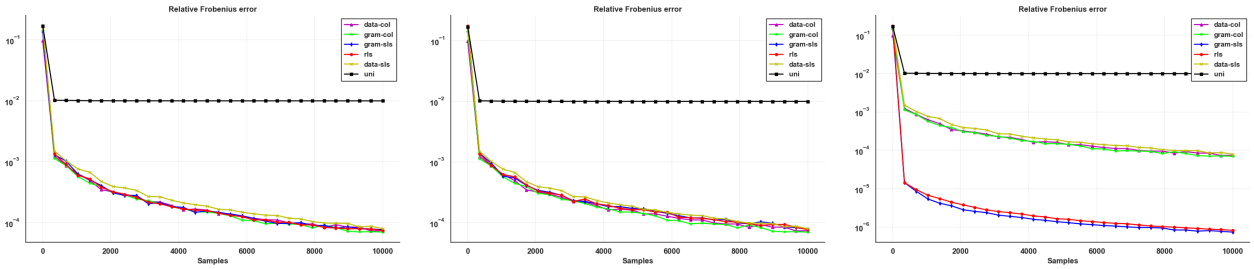
(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 20. Ridge Leverage scores for the wine data set.

A.4. **Nystrom Errors.**

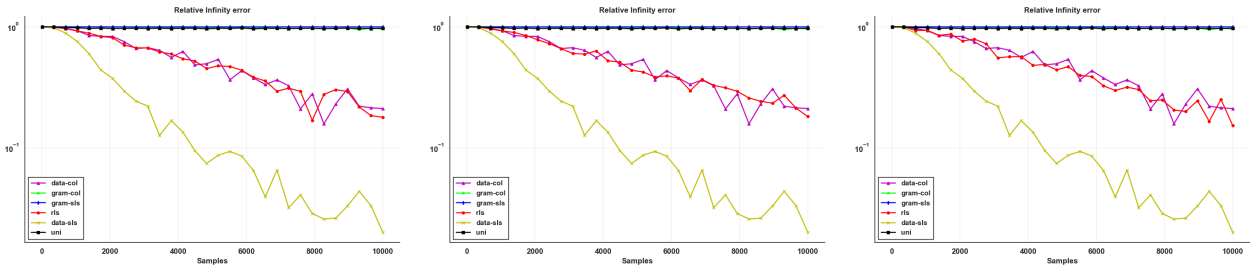


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

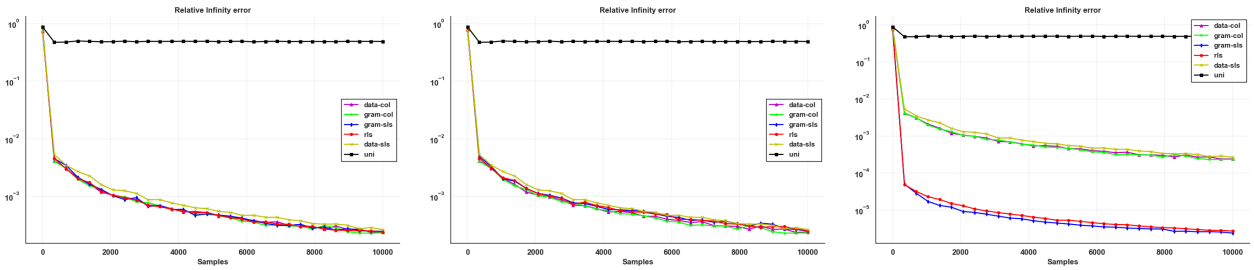


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 21. Nystrom Frobenius errors for the 3DSN data set.

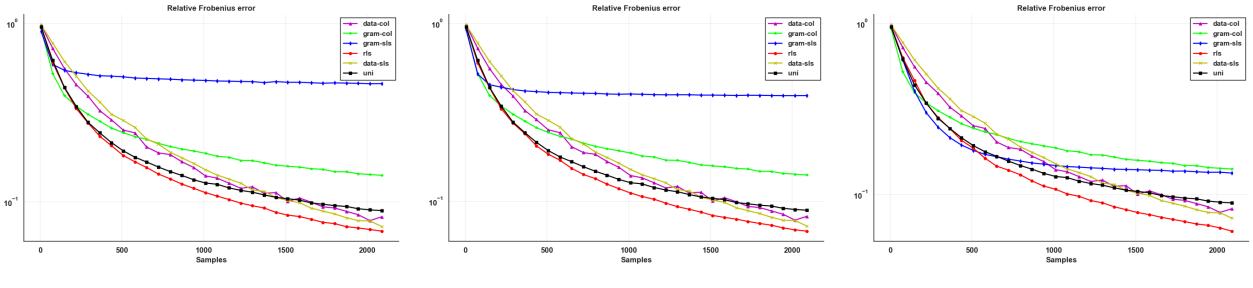


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 40$

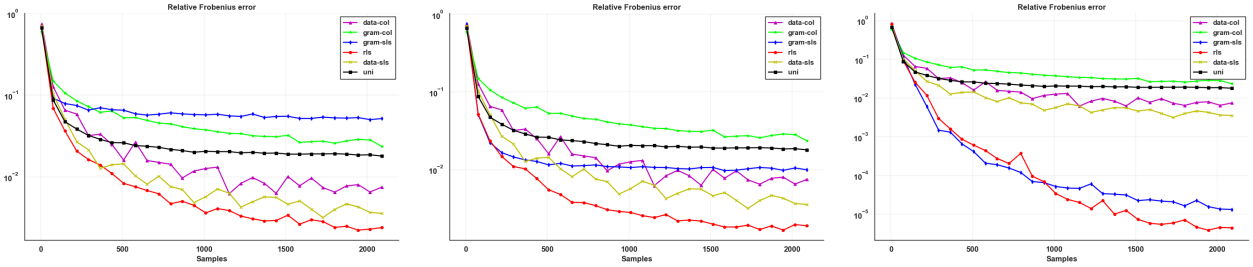


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 22. Nystrom infinity errors for the 3DSN data set.

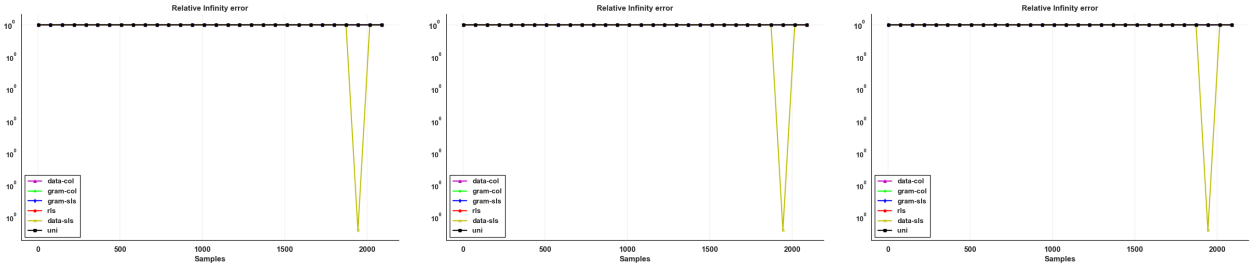


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

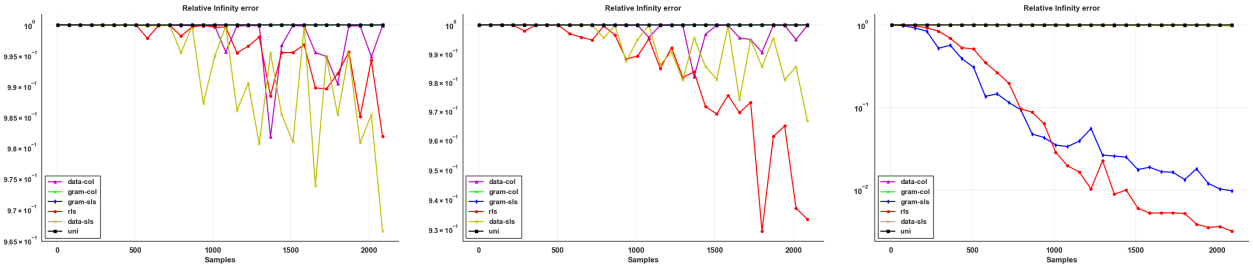


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 23. Nystrom Frobenius errors for the Abalone data set.

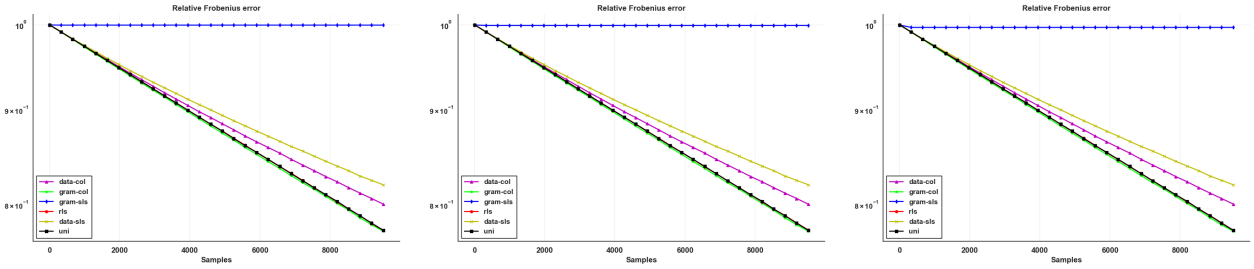


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

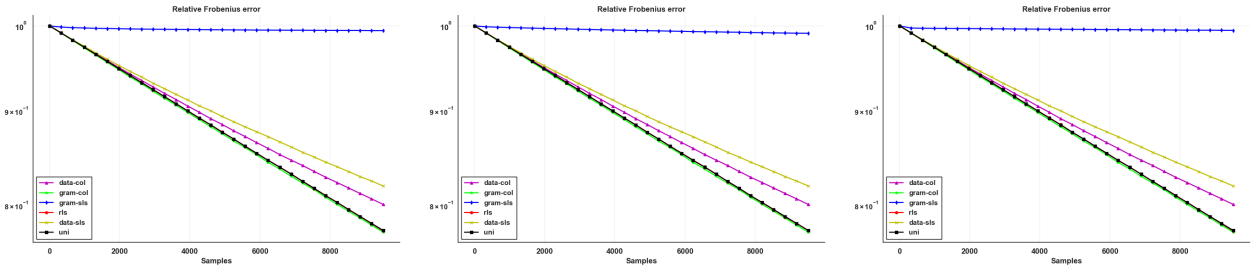


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 24. Nystrom infinity errors for the Abalone data set.

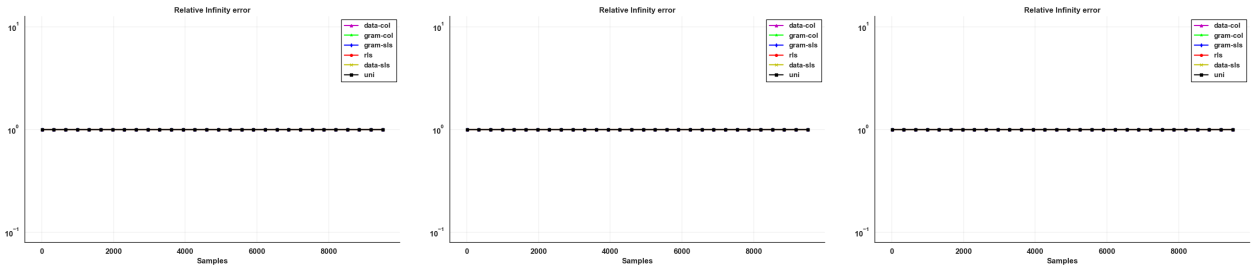


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

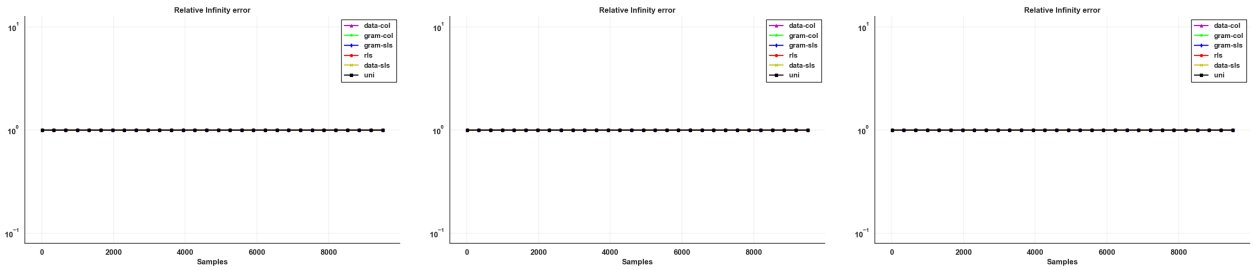


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 25. Nystrom Frobenius errors for the Magic data set.

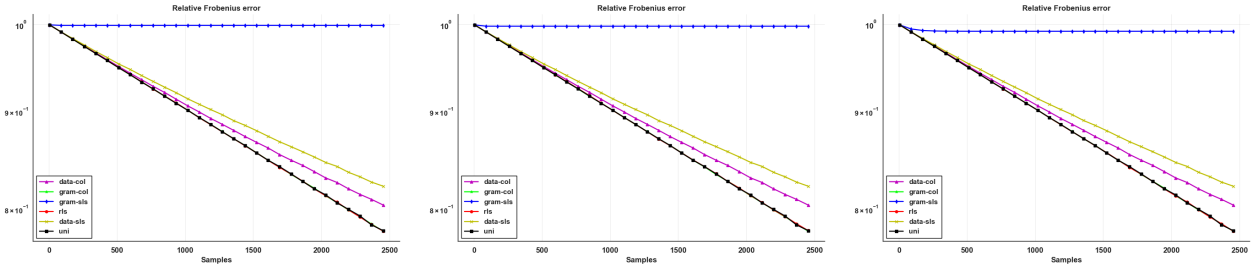


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

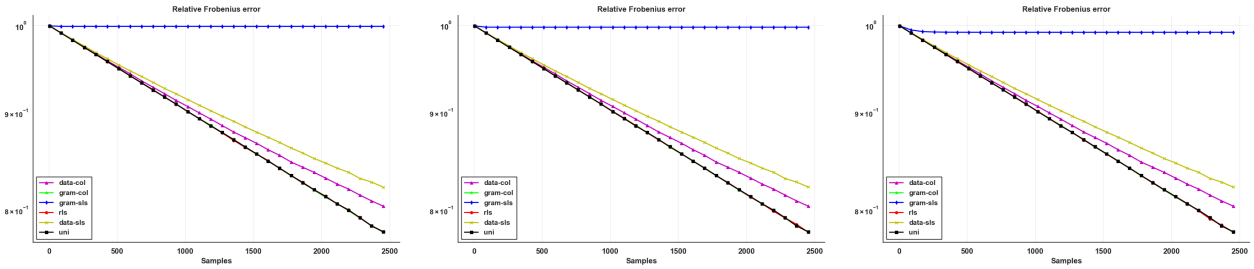


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 26. Nystrom infinity errors for the Magic data set.

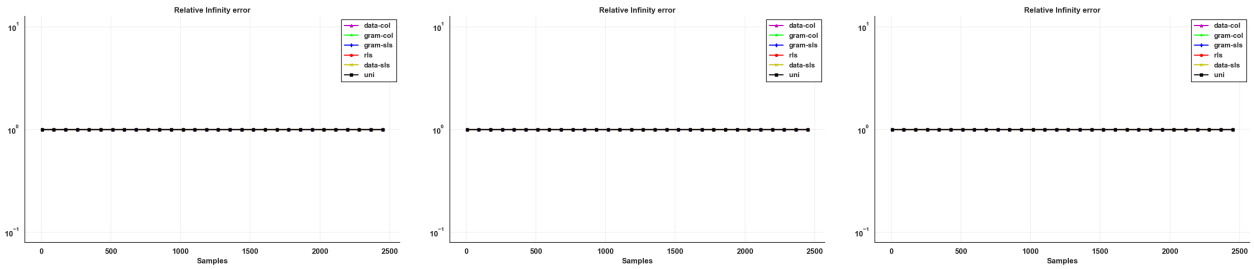


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

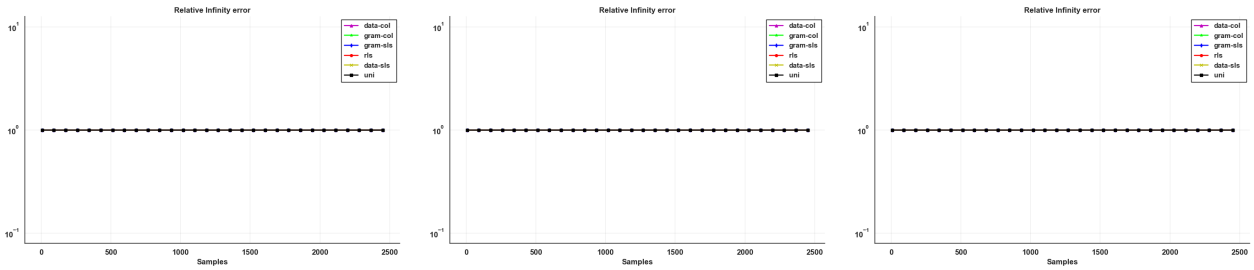


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 27. Nystrom Frobenius errors for the Stocks data set.

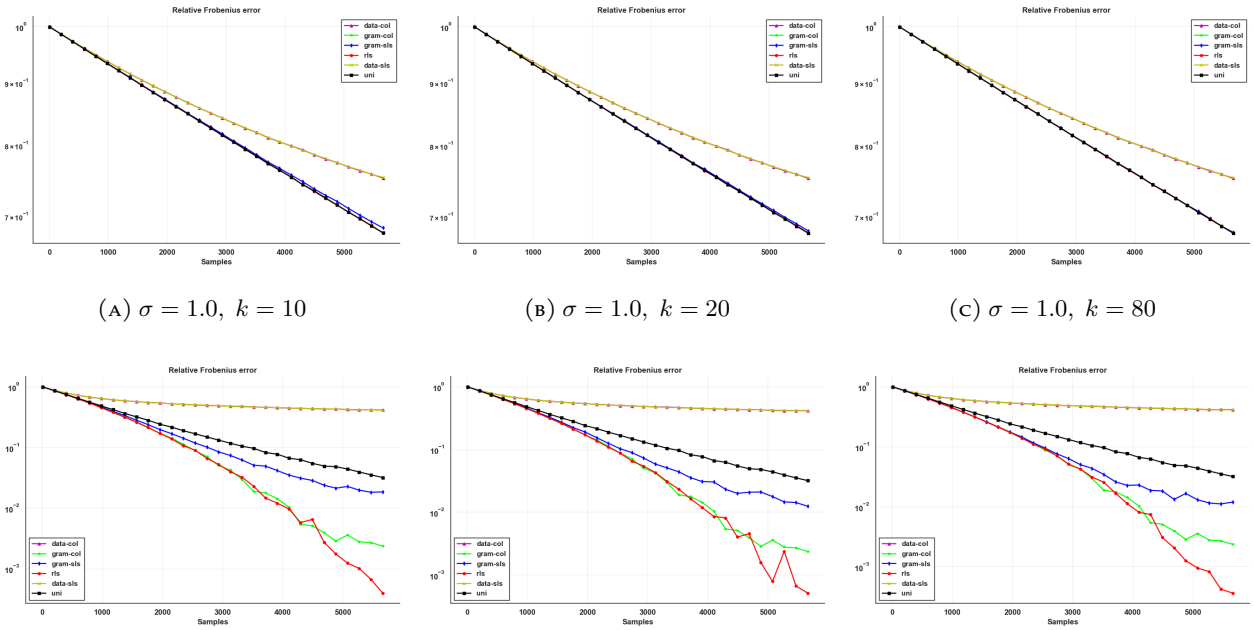


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

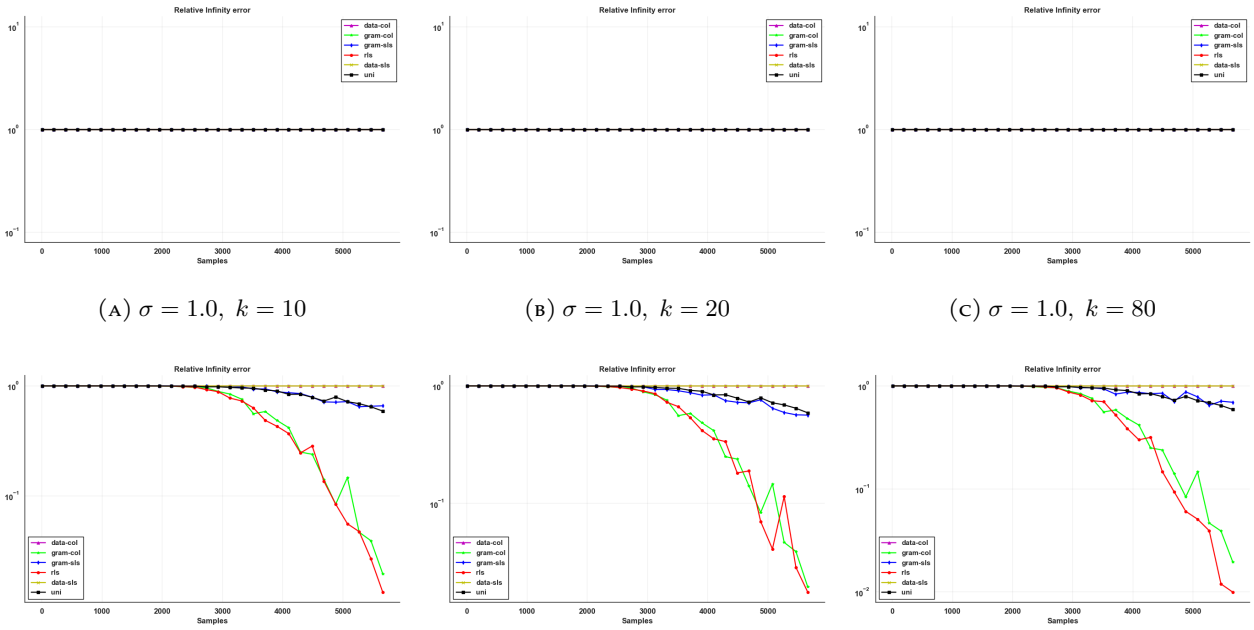
FIGURE 28. Nystrom infinity errors for the Stocks data set.



(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

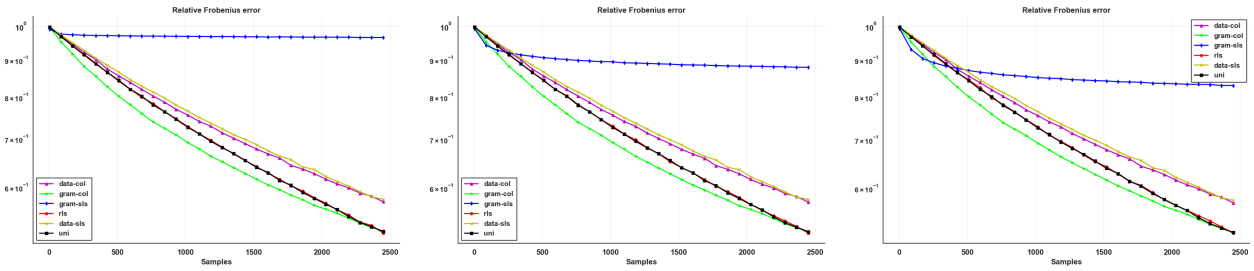
FIGURE 29. Nystrom Frobenius errors for the Temp data set.



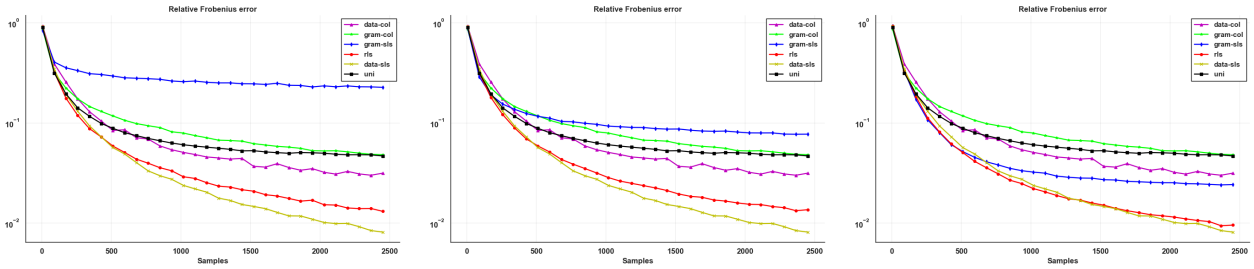
(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 30. Nystrom infinity errors for the Temp data set.

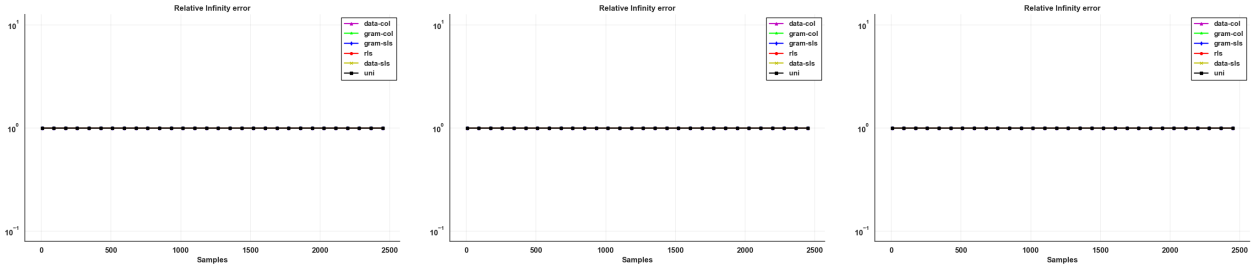


(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$

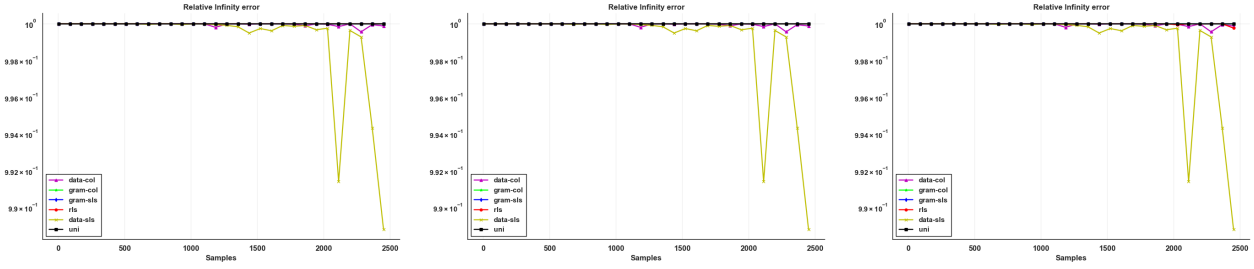


(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 31. Nystrom Frobenius errors for the wine data set.



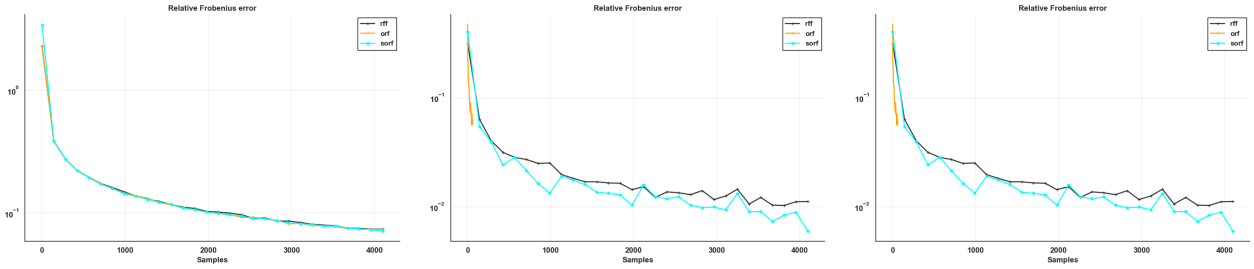
(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$



(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

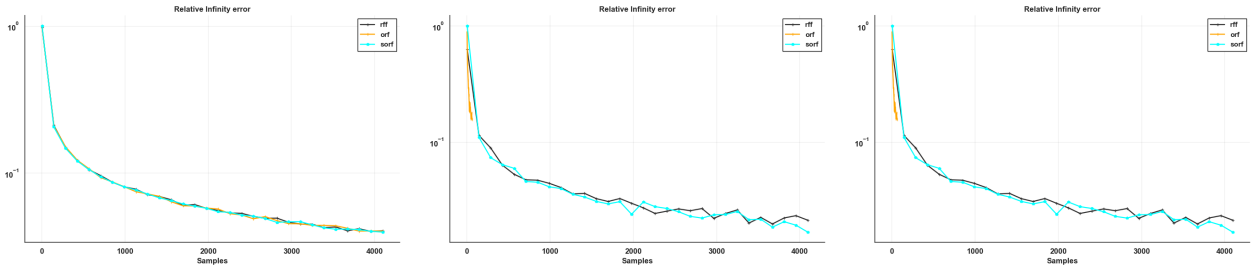
FIGURE 32. Nystrom infinity errors for the wine data set.

A.5. **Nystrom Errors.**



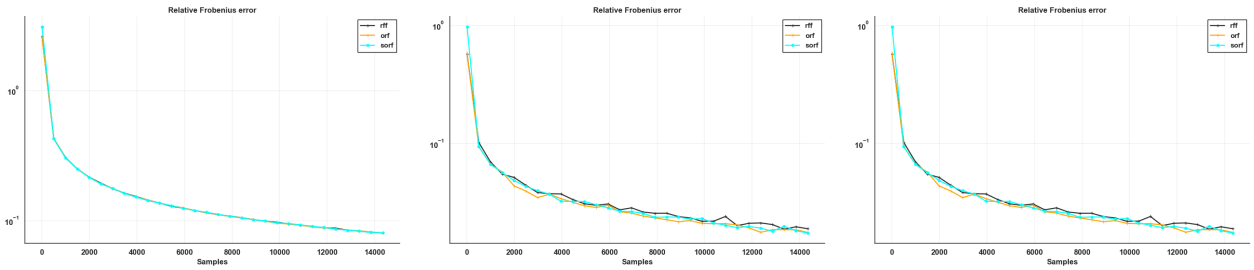
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 33. RFF Frobenius errors for the 3DSN data set.



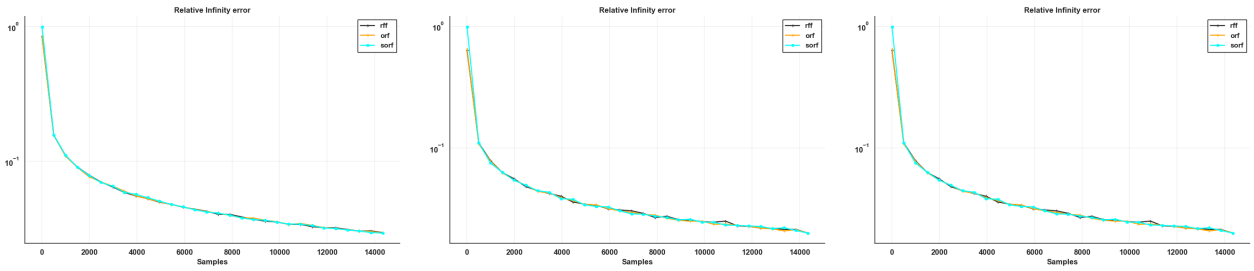
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 34. RFF absolute errors for the 3DSN data set.



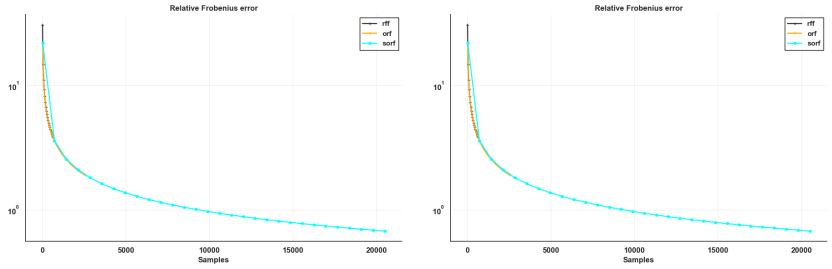
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 35. RFF Frobenius errors for the Abalone data set.



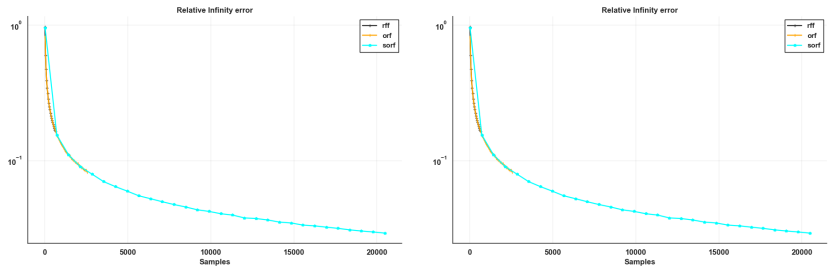
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 36. RFF infinity errors for the Abalone data set.



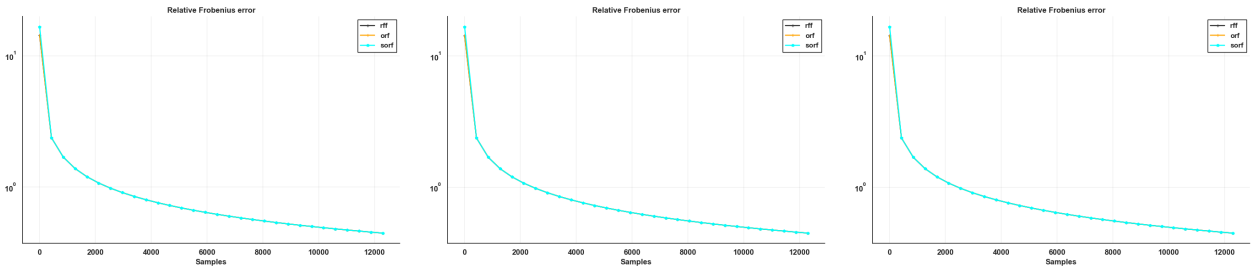
(A) $\sigma = 1.0$ (B) $\sigma = 10.0$

FIGURE 37. RFF Frobenius errors for the magic04 data set.



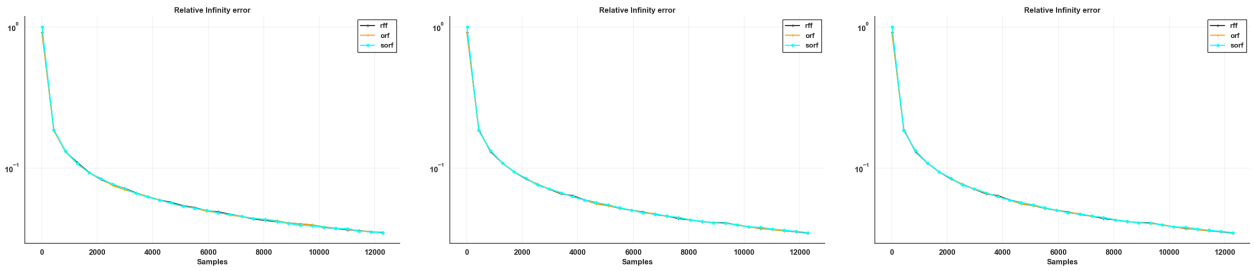
(A) $\sigma = 1.0$ (B) $\sigma = 10.0$

FIGURE 38. RFF infinity errors for the magic04 data set.



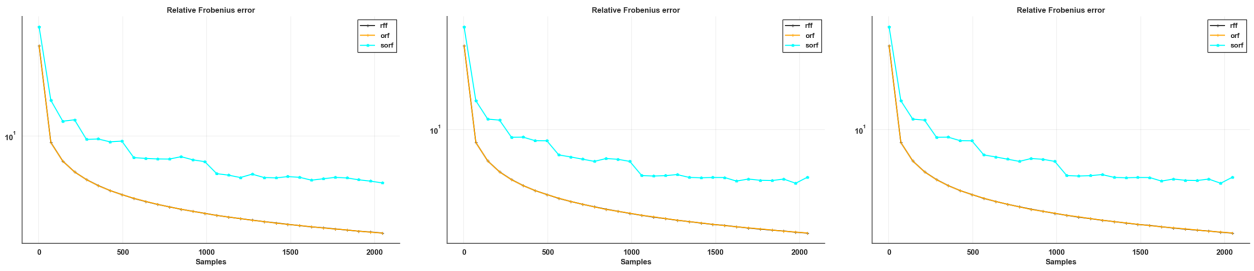
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 39. RFF Frobenius errors for the Stocks data set.



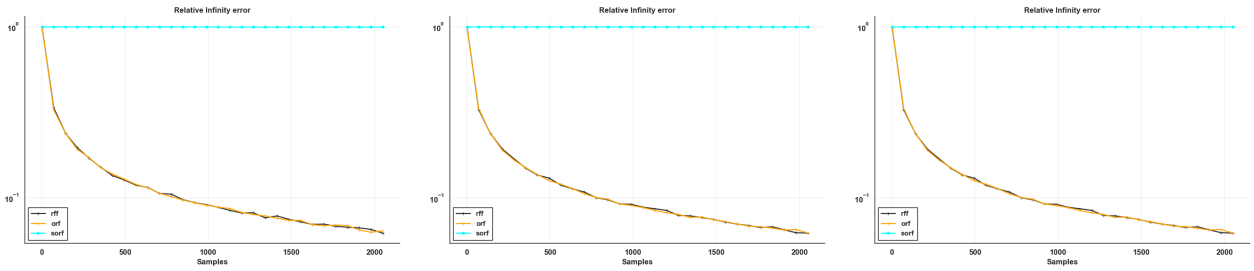
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 40. RFF infinity errors for the Stocks data set.



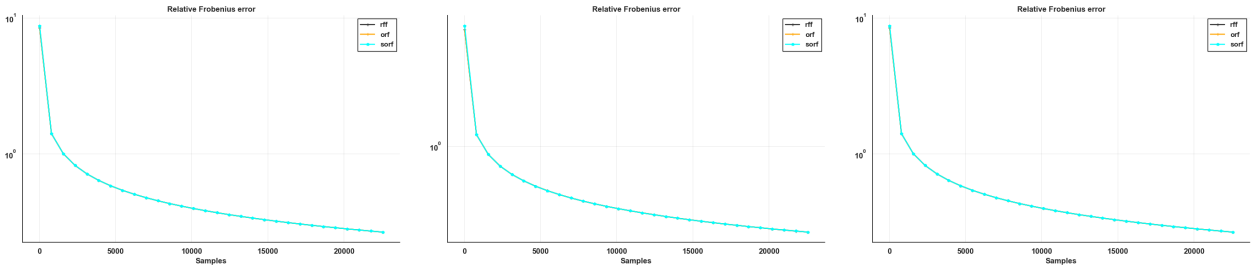
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 41. RFF Frobenius errors for the Temp data set.



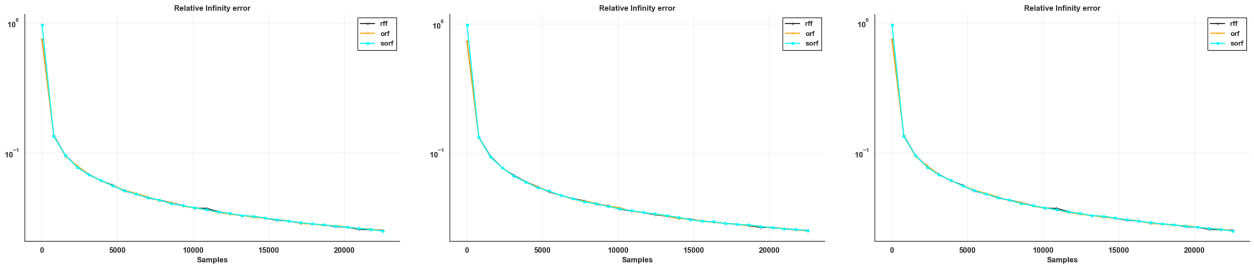
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 42. RFF infinity errors for the Temp data set.



(A) $\sigma = 1.0$ (B) $\sigma = 2.1$ (C) $\sigma = 10.0$

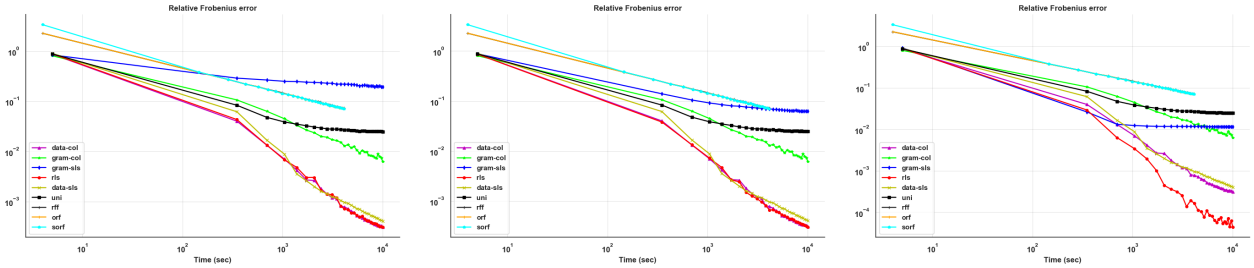
FIGURE 43. RFF Frobenius errors for the Wine data set.



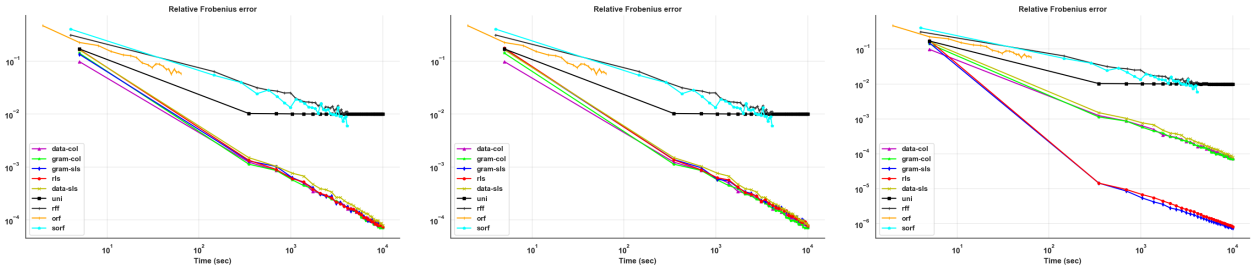
(A) $\sigma = 1.0$ (B) $\sigma = 2.1$ (C) $\sigma = 10.0$

FIGURE 44. RFF infinity errors for the Wine data set.

A.6. **Kernel Comparisons.**

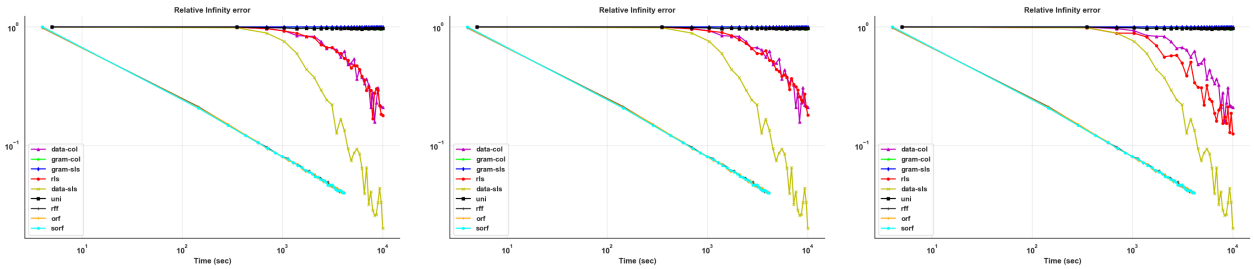


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

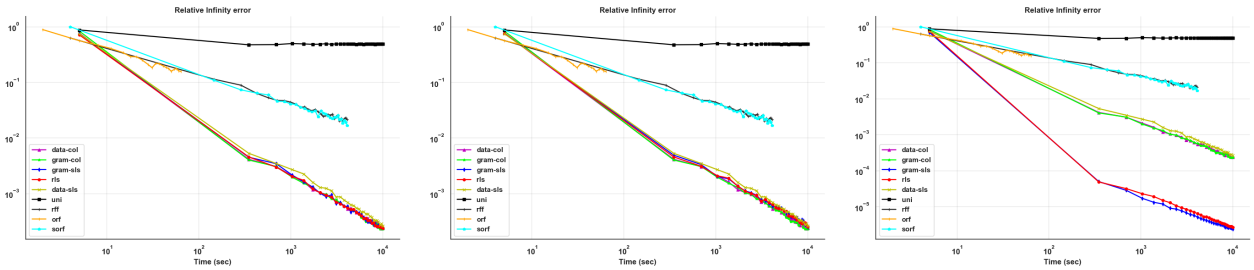


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 45. Comparing Frobenius errors of kernel methods for the 3DSN data set.

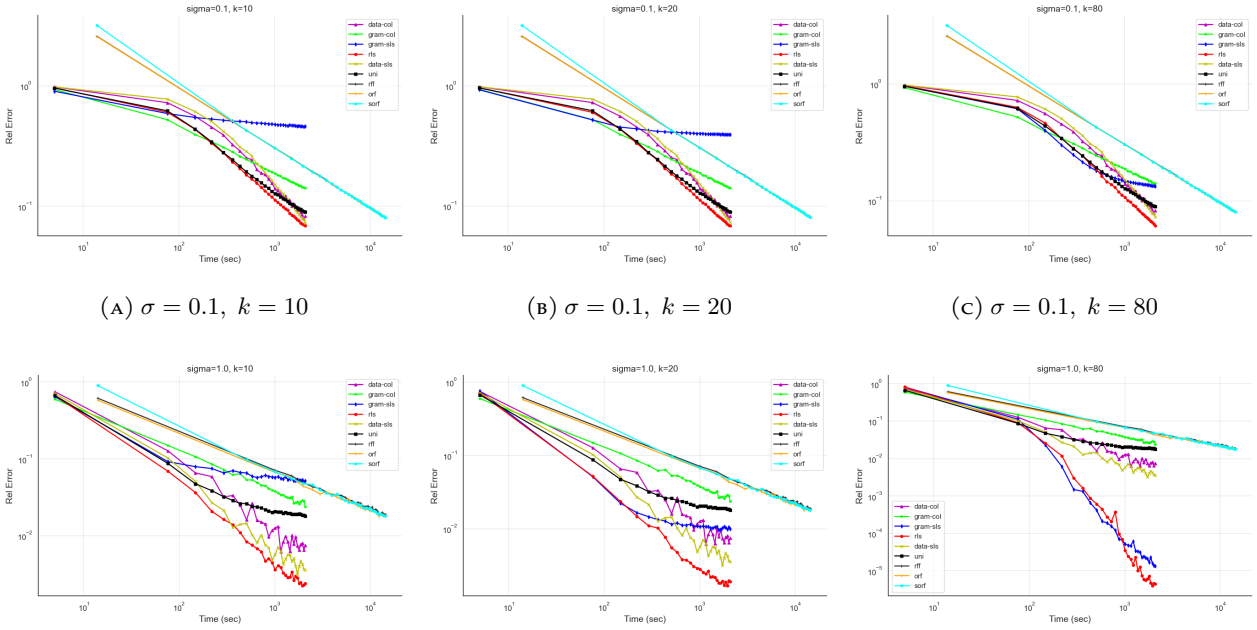


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

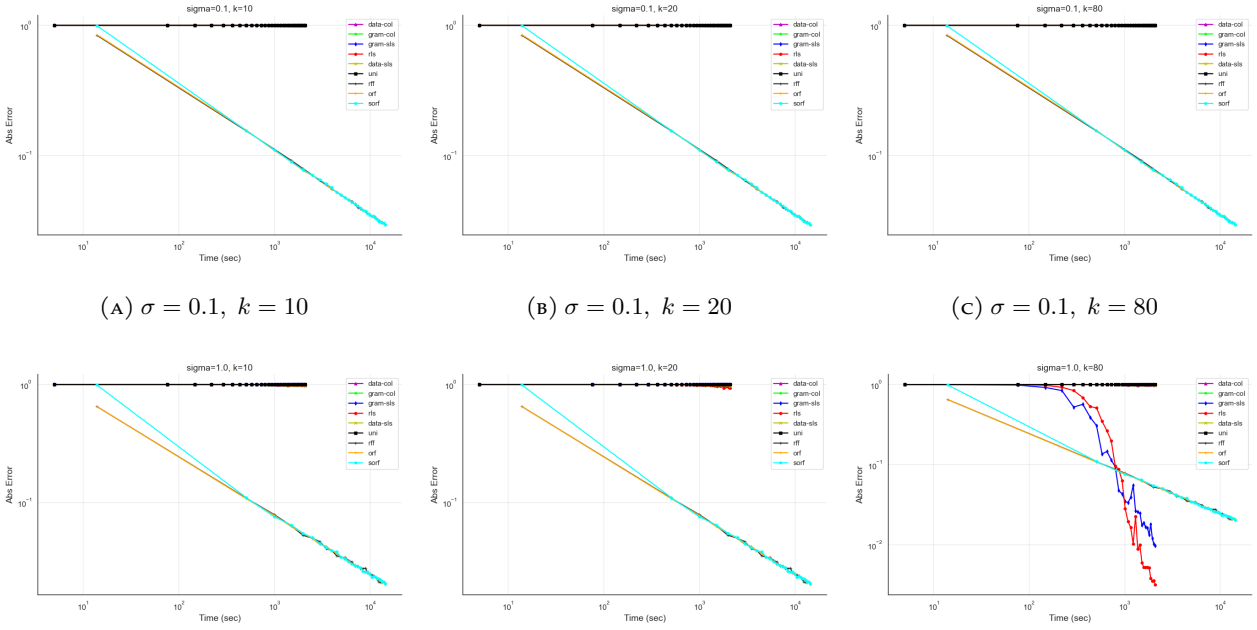
FIGURE 46. Comparing infinity errors of kernel methods for the 3DSN data set.



(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

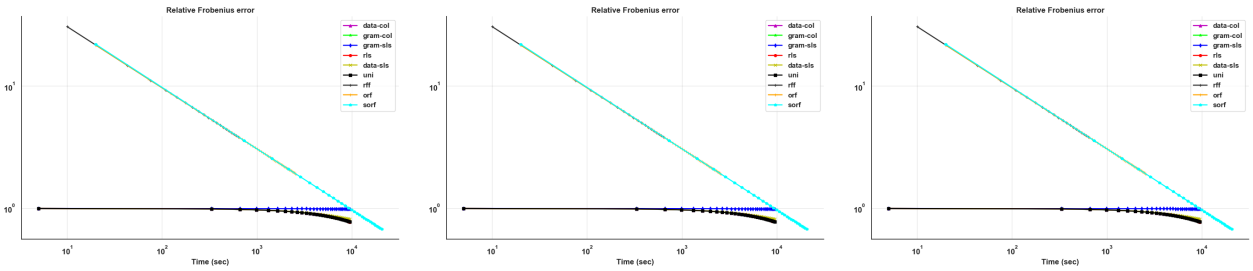
FIGURE 47. Comparing Frobenius errors of kernel methods for the Abalone data set.



(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

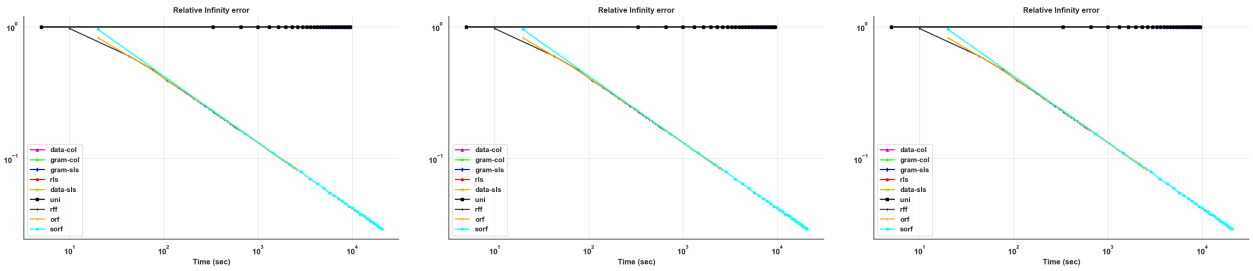
(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 48. Comparing infinity errors of kernel methods for the Magic data set.



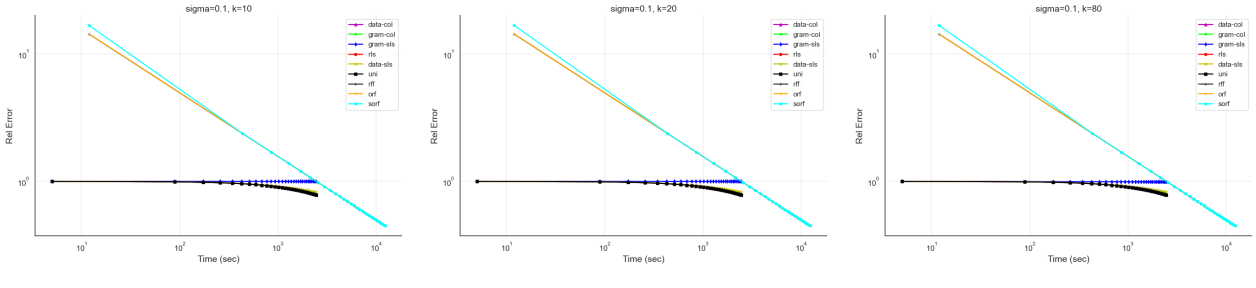
(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

FIGURE 49. Comparing Frobenius errors of kernel methods for the Magic data set.

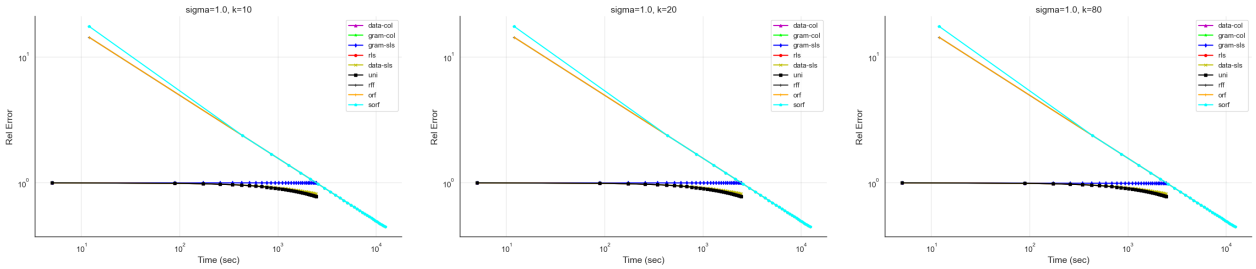


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

FIGURE 50. Comparing infinity errors of kernel methods for the Magic data set.

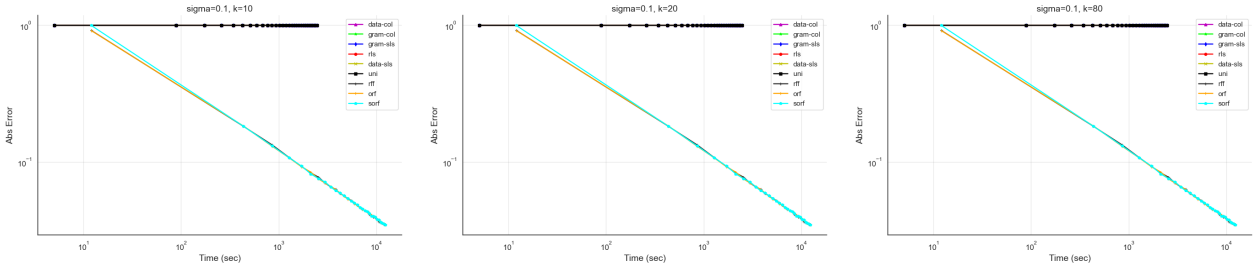


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

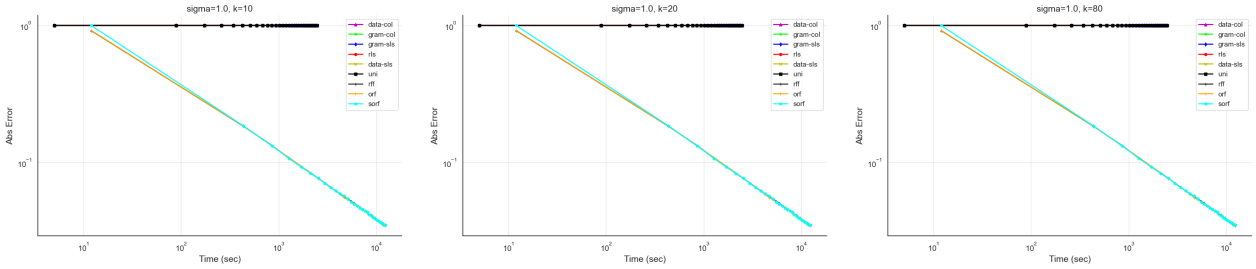


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 51. Comparing Frobenius errors of kernel methods for the Stocks data set.

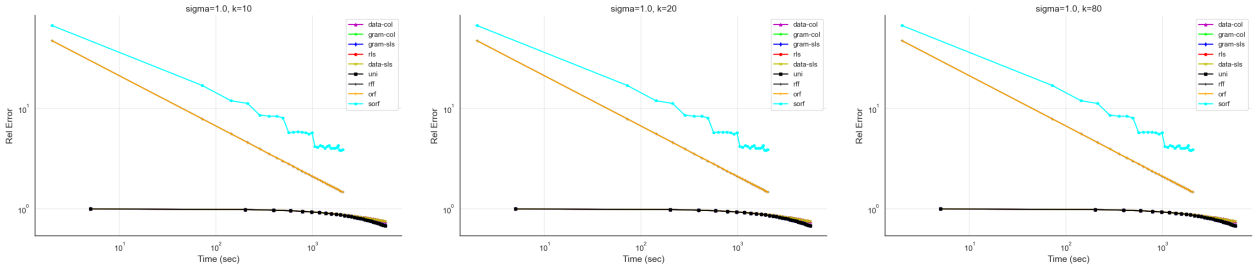


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

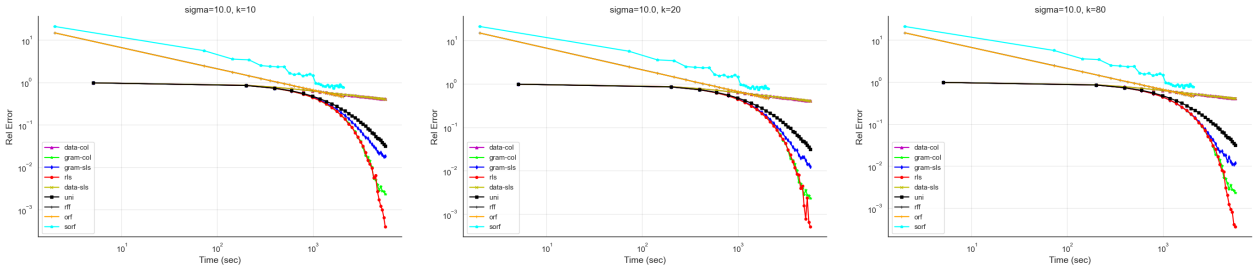


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 52. Comparing infinity errors of kernel methods for the Stocks data set.

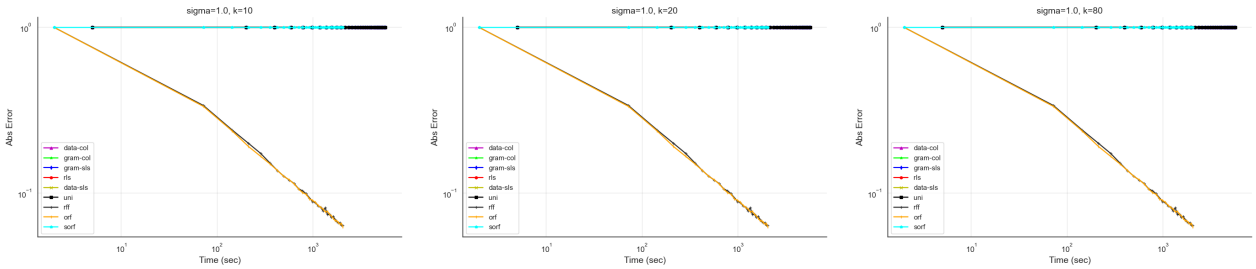


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

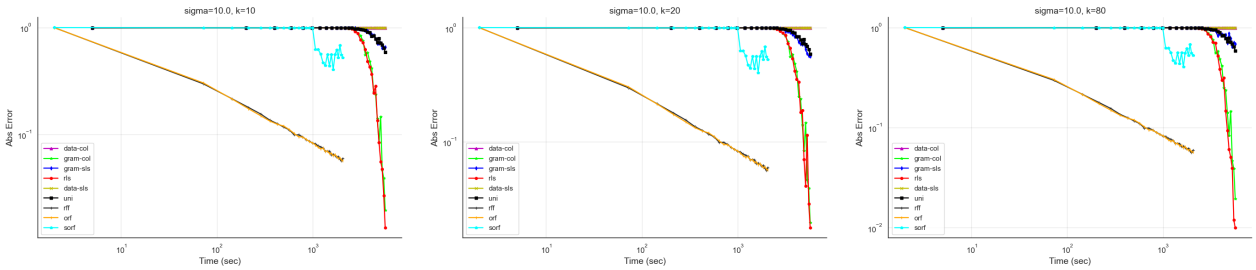


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 53. Comparing Frobenius errors of kernel methods for the Temp data set.

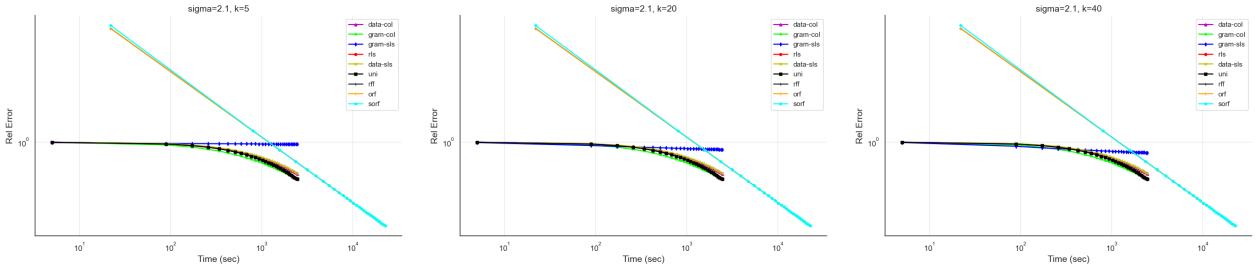


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

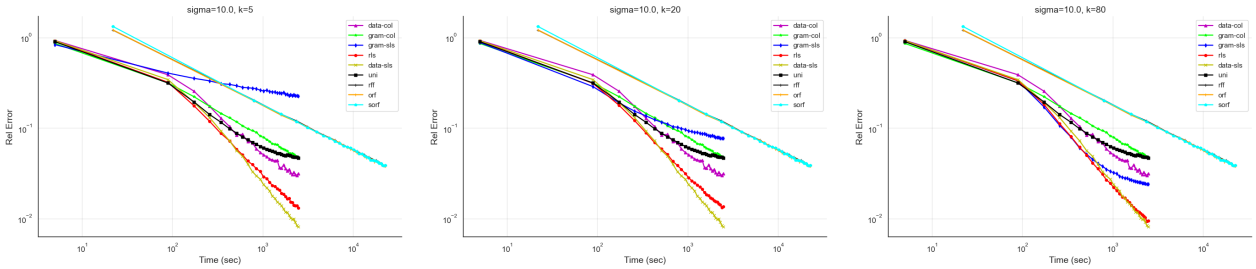


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 54. Comparing infinity errors of kernel methods for the Temp data set.

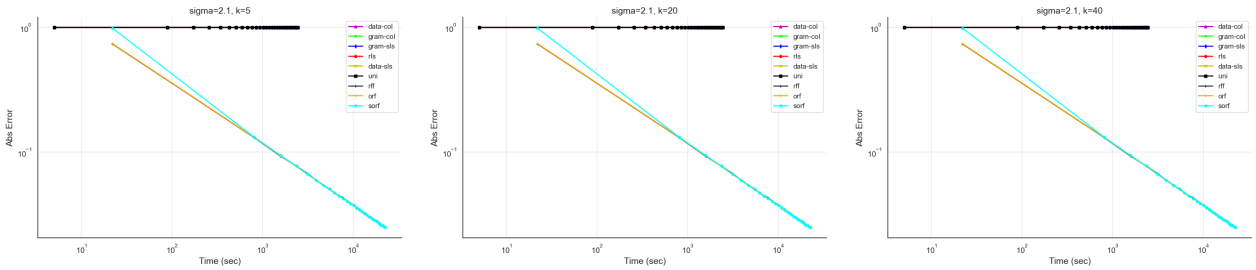


(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$

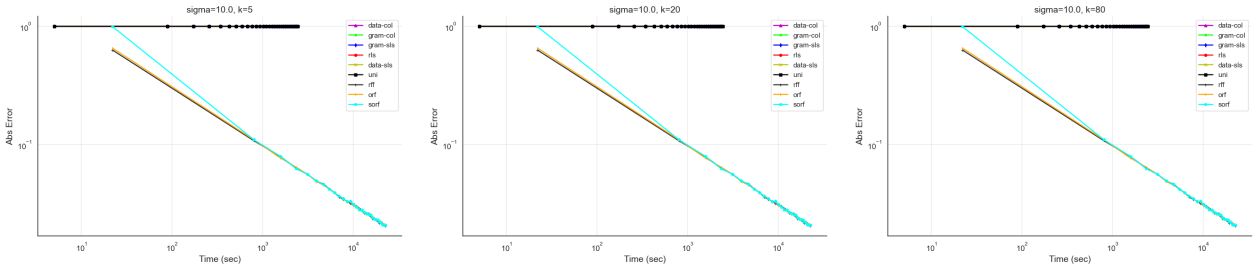


(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 55. Comparing Frobenius errors of kernel methods for the wine data set.



(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$



(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 56. Comparing infinity errors of kernel methods for the wine data set.