GRIGORIEV_ZHENYA_1818380.pdf

by Zhenya Grigoriev

Submission date: 06-Sep-2022 03:43PM (UTC+0100)
Submission ID: 185749704
File name: GRIGORIEV_ZHENYA_1818380.pdf (2.51M)
Word count: 18209
Character count: 86655

Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

General Signature Kernel For Time Series Modelling

Author:
Zhenya Grigoriev

Supervisor:
Thomas Cass

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2020-2022

September 2022

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Abstract

The dissertation presents a study on the application of the general signature kernel [CLX21] to modelling financial time series data. We conduct a thorough review of the concept, design a practical model and carry out a series of experiments to validate the proposed approach and demonstrate its advantages. Furthermore, we explore the idea of static kernel approximation applied to the proposed model to achieve better computational efficiency and discuss its advantages and shortcomings. The empirical results have been collected by running the experiments on the popular time series classification benchmark UEA datasets [Bag+18], as well as on the Bitcoin time series and technical indicators data studied earlier by [Mud+20]. The proposed general signature model implementation outperformed the baseline in most of the cases, whereas the same model with static kernel approximation showed adequate performance while achieving significant data dimensionality reduction.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Forecasting financial data has been proven difficult and the world of cryptoassets is no exception. There have been multiple attempts to do this and there is no single correct approach. Some researchers create the case for using the fundamental pricing theory to do this [Gba+21], [Det+18], [Det+21], while others argue for using the traditional quantitative finance modelling [SKA21], [CF19]. Finally, there is a large group of researchers who employ Machine Learning (ML) to the task of cryptocurrency price forecasting [AG19], [JL18].

The focus of this thesis lies in the third type of forecasting methodologies, in particular we will be looking at the generalisation power of Support Vector Machine (SVM) model in classification and prediction tasks. A valid question to ask is if there have been any previous successful attempts to apply SVMs to learning financial data. Indeed there are examples in electricity markets [Shi+15], commodities [Xie+06], currency markets (FX) [USC07] and, of course, equities [CLN12]. In fact, to validate the proposed model empirically, we compare it against the results of the paper [Mud+20] which surveys different machine learning models for the task of learning cryptocurrency data.

SVM is an example of the so-called ”Kernel method” where a kernel function is designed in a way to exploit the information that can be gained by mapping the input space to a potentially infinite-dimensional space without explicitly computing this mapping. This creates an opportunity for promising research in the area of appropriate mappings and their inner products for different kinds of data, such as graphs [Vis+08], text [Mar+11] or even video streams [ZPC11]. Another promising direction in the area of machine learning is application of the Signature feature mapping. In their celebrated paper [LLN16] demonstrate the benefits of using the path signature for the regression problem where the input is a stream of data. The same approach has also been applied to financial data [Gyu+14], and, more recently, used with the powerful concept of kernel machine learning via the signature kernel [Sal+21].

Kernel methods also have their downsides - great expressive power of the model comes at a high computational cost. Combined with the computational overhead to evaluate the signature kernel, it renders the methodology inapplicable for large data sets. Kernel methods approximation is an active field of research, see, for example [Wan+15], [Liu+21], and it seems possible that methods proposed in the literature

will be applicable to the signature kernel setup as well.
The goals of this work are two-fold: firstly, we would like to introduce the reader to the novel concept of the general signature kernel [CLX21] and walk through one of the ways to implement it in practice. Moreover, we provide a systematic comparison between the general signature kernel and the less flexible signature-PDE kernel discussed in [Sal+21] using the real-world data, as well as demonstrate the benefits of using the signature kernel learning on the dataset of Bitcoin prices and other technical indicators. We also discuss the shortcomings of the signature kernel approach and explore the directions that can be taken to alleviate them.

In what follows we provide a brief technical description of the methods (chapter 2), which we use to discuss the model design and components (chapter 3) and, finally, demonstrate the competitive advantage of the proposed solution via experiments using the benchmark dataset (chapter 4).

Chapter 2

Theoretical foundations

2.1 Supervised learning

In a supervised learning problem, we are provided with pairs of samples $(\boldsymbol{x}, y)$ where $\boldsymbol{x} \in \mathbb{R}^D$ are *observations* that are used to predict a *target variable* y[1]. When the target variable $y \in \mathcal{S}$ where $\mathcal{S}$ is a finite set, we call this a *classification* task (binary classification when $|\mathcal{S}| = 2$), whereas the case of $y \in \mathbb{R}$ is referred to as a *regression* task. The goal is to come up with a model $f_\theta(\boldsymbol{x})$ which would accurately represent the relationship between the observations and the target variable. Such model usually depends on some parameters θ, and we are given a set of N training examples $\{ \boldsymbol{x_i}, y_i \}, i = 1, ..., N$ that are used to calibrate these parameters so as to make predictions $f_\theta(\boldsymbol{x}_i)$ as close as possible to ground truth y_i (known as *learning*). Once calibrated, the model can be used to estimate the value of the target variable $\hat{y} = f_\theta(x')$ for observations x' outside the training dataset (known as *inference*).

2.1.1 A brief introduction to linear classifiers

Consider the simplest binary classification problem depicted in fig. 2.1, where the classes of points are clearly separable by a line (hyperplane for more than 2 dimensions). In such cases we say that the classes are linearly separable and can define the hyperplane that separates them via $H = \{ \boldsymbol{x} \in \mathbb{R}^D : \boldsymbol{w}^\top \boldsymbol{x} + b = 0 \}$. Given such a hyperplane H parameterised with $\boldsymbol{w}$ and b we can define the simplest classifier

$$f(x) = sign(w^\top x + b) \tag{2.1}$$

If we label classes as $y_i \in \{ -1, 1 \}$, the classifier is correct when $f(x_i)$ and y_i have the same sign. Considering the example in fig. 2.1 again, intuitively, the orange line separates the points "better" than the blue one. Although they are equally good on the training set, we may hypothesise that the blue line would perform worse on the unseen examples as it is skewed towards one of the classes. This motivates us to introduce another measure of "goodness" of a classifier. Let the functional margin of an example (x_i, y_i) be $\gamma_i = y_i(w^\top x_i + b)$, which implies that classification is correct

[1] To fix the notation, we use **bold** vectors unless this is obvious from the context.

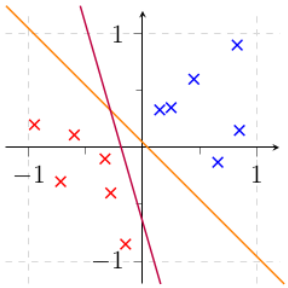


Figure 2.1: The example of a binary classification problem, where the labels $y_i \in \{Blue, Red\}$ and $\boldsymbol{x_i} = (x_1, x_2), x_{ij} \in [-1, 1]$. The two classes are linearly separable, possible separating lines are shown in orange and purple.

if $\gamma_i > 0$. If we normalise the functional margin by the magnitude of w, we obtain the geometric distance $d(H, x_i) = \gamma_i / ||w||$ from the hyperplane to the data point. Our goal is to find a hyperplane which splits the training set "somewhere in the middle", i.e. maximises the geometric distance from all the points in the training set (or, equivalently, minimises $||w||$ subject to classifying the points correctly). We can formulate this as an optimisation problem

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \boldsymbol{w}^\top \boldsymbol{w} \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 \\ & i = 1, ..., N \end{aligned} \tag{2.2}$$

Using the method of Lagrange multipliers [CS08] the problem can be transformed into its corresponding dual

$$\begin{aligned} \max \quad & W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ & i = 1, ..., N, \alpha_i \geq 0 \end{aligned} \tag{2.3}$$

By solving (2.3) for the optimal $\boldsymbol{\alpha}$ and finding $\boldsymbol{w}$ and b from the primal constraints, we get the optimal hyperplane parameterised by $(\boldsymbol{w}^*, b^*)$.

Definition 2.1.1 (Support vectors [CS00])
The optimal solution must satisfy

$$\alpha_i^*(y_i(\langle w^*, x_i \rangle + b^*) - 1) = 0, \quad \alpha_i^* \geq 0$$

which implies that when the functional distance γ_i of a sample $\boldsymbol{x}_i$ is greater than 1 then $\alpha_i = 0$, meaning that only the closest data points to the hyperplane contribute to the solution $\boldsymbol{w}_i$. These data points $(\boldsymbol{x_i}, y_i)$ for which $\alpha_i > 0$ are called support vectors.

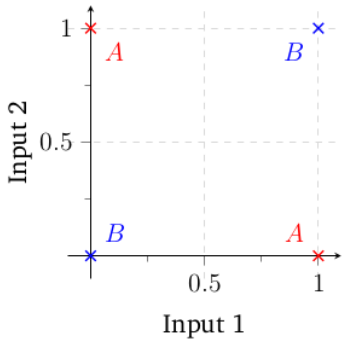


Figure 2.2: The XOR problem showing that classes A and B cannot be separated by a straight line.

Note, it is not strictly necessary to transform the optimisation problem (2.2) to its dual (2.3) since we could solve (2.2) immediately. However as we shall see later, the key reason for this is that solution of the dual (2.3) only depends on the input data via the inner product $\langle x_i, x_j \rangle$, which will help us generalise the method to non-linear problems.

2.1.2 Limitations of linear models

Consider the classical example in fig. 2.2 which demonstrates the shortcomings of linear models introduced above.
The input data space is

$$X = \{ (x_1, x_2) : x_1, x_2 \in \{ 0, 1 \} \} = \{ 0, 1 \}^2$$

which we would like to separate into classes $Y = \{ A, B \}$. Whilst it is clearly impossible to find a straight line which separates the classes A and B, does this simple example invalidate the classifier introduced above? Not necessarily. If we could find a mapping $\phi(\boldsymbol{x})$ from the input space X to a higher-dimensional *feature space* $\mathcal{H}$ where the data points could be separated by a hyperplane, we could still use the developed methods.
In the next chapter we would like to explore the properties of the feature space $\mathcal{H}$, different feature mappings $\phi(\boldsymbol{x})$ and how this affects the optimisation problem (2.3). In particular, the relationship between $\langle x_i, x_j \rangle$ and its feature space adaptation $\langle \phi(x_i), \phi(x_j) \rangle$.

2.2 Kernel methods

In this section we are going to outline the basic properties of the feature space, introduce the notion of a kernel function and connect it with the inner product of two feature mappings $\langle \phi(x_i), \phi(x_j) \rangle$ discussed in the previous chapter. Specifically,

we shall see how complex feature mappings $\phi(\cdot)$ can be implicitly used in learning algorithms via the use of corresponding kernel functions. In what follows we define the *input space* $\mathcal{X}$ to be a subset of $\mathbb{R}^n$ for simplicity, however presented results can easily be extended to $\mathbb{C}^n$ [SS].

2.2.1 Key definitions

First we would like to introduce a few important concepts related to the feature space $\mathcal{H}$. We mostly follow [CS08] for definitions, however there is extensive literature on kernel methods in machine learning, e.g. [SS], [SC04].

Definition 2.2.1 (Inner product)
Let $\mathcal{H}$ be a vector space over $\mathbb{R}$. A function $\langle \cdot, \cdot \rangle_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \mapsto \mathbb{R}$ is called inner product on $\mathcal{H}$ if

- $\langle \alpha_1 f_1 + \alpha_2 f_2, g \rangle_{\mathcal{H}} = \alpha_1 \langle f_1, g \rangle_{\mathcal{H}} + \alpha_2 \langle f_2, g \rangle_{\mathcal{H}}$
- $\langle f, g \rangle_{\mathcal{H}} = \langle g, f \rangle_{\mathcal{H}}$
- $\langle f, f \rangle_{\mathcal{H}} \geq 0$ *and* $\langle f, f \rangle_{\mathcal{H}} = 0$*, if and only if* $f = 0$

Definition 2.2.2
The norm can be defined using the inner product: $||f||_{\mathcal{H}} := \sqrt{\langle f, f \rangle_{\mathcal{H}}}$

Definition 2.2.3 (Hilbert space)
A Hilbert space $\mathcal{H}$ is a vector space endowed with the inner product with additional properties

- *(Complete) Every Cauchy sequence $\{ h_n \}_{n \geq 1}$ of elements of $\mathcal{H}$ converges to an element $h \in \mathcal{H}$, where Cauchy sequence is such that*

$$\sup_{m \geq n} ||h_n - h_m|| \to 0, \text{ as } n \to \infty$$

- *(Separable) For any $\epsilon > 0$ there is a finite set of elements $h_1, \ldots, h_N$ of $\mathcal{H}$ such that for all $h \in \mathcal{H}$*

$$\min_i ||h_i - h|| < \epsilon$$

For the purposes of this work we will require our feature space $\mathcal{H}$ to be a Hilbert space.

Definition 2.2.4 (Kernel)
A function operating on the input space $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a kernel if there exists a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{X} \mapsto \mathcal{H}$ such that $\forall x, x' \in \mathcal{X}$

$$k(x, x') := \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$$

Example 2.2.1
Consider a two-dimensional input space $\mathcal{X} \subset \mathbb{R}^2$ and the following feature map

$$\forall \boldsymbol{x} = (x_1, x_2) \in \mathcal{X}, \phi : \boldsymbol{x} \mapsto \boldsymbol{\phi(x)} = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \in R^3 = \mathcal{H}$$

The feature maps the inputs in such that linear terms in the input space become quadratic in the feature space. The inner product in the feature space can then be evaluated as follows:

$$\begin{aligned}\langle \boldsymbol{\phi(x)}, \boldsymbol{\phi(z)} \rangle &= \langle (x^2, \hat{x}^2, \sqrt{2}x\hat{x}), (z^2, \hat{z}^2, \sqrt{2}z\hat{z}) \rangle \\ &= x^2z^2 + \hat{x}^2\hat{z}^2 + 2x\hat{x}z\hat{z} = (xz + \hat{x}\hat{z})^2 \\ &= \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2\end{aligned}$$

We call the function $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2$ a kernel function with the corresponding feature map ϕ and feature space $\mathcal{H}$.

Note that in this example we show it is possible to evaluate the inner product between two vectors in the feature space $\langle \boldsymbol{\phi(x)}, \boldsymbol{\phi(z)} \rangle$ without explicitly computing the transformation $\phi(\cdot)$. Later, we consider examples where the feature space $\mathcal{H}$ is infinite dimensional, making it impossible to perform the transformation from the input space $\mathcal{X}$ to the feature space $\mathcal{H}$ explicitly. The kernel function is very important in this case as it allows us to compute the inner product of two feature vectors even in such spaces.

2.2.2 Properties of a kernel function

One reasonable question to ask is, given a kernel function $k(\cdot)$ is it possible to find the feature space $\mathcal{H}$ such that the evaluation of the kernel would match the evaluation of the inner product in this feature space?

Definition 2.2.5 (Gram matrix)
Consider the inputs $x_1, ..., x_n \in \mathcal{X} \subset \mathbb{R}$ and a function $k : \mathcal{X}^2 \mapsto \mathbb{R}$, the Gram matrix is the $n \times n$ matrix $\boldsymbol{G}$ with elements

$$G_{i,j} = k(x_i, x_j)$$

The function $k(\cdot)$ is called a kernel function and the matrix $\boldsymbol{G}$ is referred to as a kernel matrix.

Definition 2.2.6
A matrix $\boldsymbol{G} \in \mathbb{R}^{n \times n}$ is called positive semi-definite if $\forall \boldsymbol{c} \in \mathbb{R}^n$

$$\boldsymbol{c}^\top \boldsymbol{G} \boldsymbol{c} = \sum_{i,j} c_i c_j G_{i,j} \geq 0$$

The kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called positive semi-definite if it is symmetric and the corresponding Gram (kernel) matrix formed by restriction of the domain to any finite subset of the space $\mathcal{X}$ is positive semi-definite.

The following theorem connects positive semi-definite functions and kernel functions.

Theorem 2.2.1
Any kernel function defined as in 2.2.4 is positive semi-definite.

Proof See Appendix A.1

In [2.2.1] we presented one example of a kernel function and have just shown how kernel functions and symmetric positive semi-definite matrices are related. A natural question to ask is, how could we construct expressive kernels for use in complex real-world problems? It turns out, we can compose simple kernel functions to obtain more complex ones. Below we detail some of the transformations that can be used, but the list is far from complete, and, essentially any transformation that preserves the property of positive semi-definiteness of the kernel matrix would be sufficient.

Theorem 2.2.2
Consider an input space $\mathcal{X}$, valid kernel functions k, k_1, k_2 on $\mathcal{X}$ and $\beta > 0 \in \mathbb{R}$, then

- *(sum of kernels) βk and $k_1 + k_2$ are valid kernels on $\mathcal{X}$*
- *(product of kernels) $k_1 \times k_2$ is a valid kernel on $\mathcal{X}$*

Proof See Appendix A.2

This remarkable fact allows us to define new kernels using combinations of positive semi-definite functions which are guaranteed to provide the inner product in a feature space $\mathcal{H}$ between features $\phi(x)$, where the feature map $\phi(\cdot)$ does not even need to be specified explicitly.

Definition 2.2.7 (Kernel trick)
Given a learning problem formulated in terms of the inner product, we can avoid explicitly computing feature mappings $\phi(\cdot)$ in high dimensions by finding an appropriate kernel function $k(\cdot, \cdot)$ which, when evaluated on members of the input space, is equivalent to the inner product of their feature mappings.

2.2.3 Examples of kernel functions

We now outline a few important examples of kernels that are used in practice and can be constructed using the ideas from the previous section.

Example 2.2.2 (Polynomial kernel)
Let $x, x' \in \mathbb{R}^n$, $d \geq 1 \in \mathbb{Z}$ and $c \geq 0 \in \mathbb{R}$. The polynomial kernel can be defined as

$$k(x, x') := (\langle x, x' \rangle + c)^d$$

Other important examples of kernels include

- (Gaussian) $k(x, x') := \exp\left(-\frac{\|x-x'\|_2^2}{\sigma^2}\right), \sigma > 0$

- (Exponential) $k(x, x') := \exp(\langle x, x' \rangle)$

So far in the discussion above we considered $\mathcal{X}$ to be a subspace of $\mathbb{R}^n$, however this is not required in general. This work is mostly concerned about kernels on paths in $\mathbb{R}^d$ (we define the notion of a *path* in later sections), but there are examples in the literature where kernels are defined on arbitrary objects, such as in the space of strings [Mar+11] or graphs [Vis+08].
A few kernel functions will be very relevant to our work. Firstly, it is the Gaussian radial basis function (Gaussian RBF) described above, which is a key building block in the final model presented in the chapter 3. Secondly, the Global Alignment Kernel (GAK) [Cut+07] is a popular baseline for any new kernel where the input space is time-series data, for example, [Sal+21] compare the performance of their method with GAK.
Thanks to the Kernel trick [2.2.7] we can now compute the inner product between feature vectors $\langle \phi(x_i), \phi(x_j) \rangle$ by evaluating a kernel function on the members of the input space itself (x_i, x_j). Next we would like to revisit the linear classifier model described in section 2.1.1 and see how it model can be "kernelised" to make decisions by computing the inner product in the feature space.

2.3 Support vector machine

One key observation for discussion in this section is that the dual optimisation problem [2.3] only depends on the inner product of the features, not the features themselves. Equipped with the ideas derived in the previous section, we aim to improve the linear model by utilizing the machinery of the Kernel trick [2.2.7].

2.3.1 Kernelised support vector machines

If the input data is not linearly separable in the input space, we can map it to a (usually higher-dimensional) feature space where it is linearly separable and compute the inner product between features. The optimisation problem eq. (2.3) can therefore be reformulated as

$$\begin{aligned} \max \quad & W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ & i = 1, ..., N, \alpha_i \geq 0 \end{aligned} \tag{2.4}$$

where a valid kernel function $k(\cdot, \cdot)$ replaces the Euclidean inner product between two elements of the input space $\langle x_i, x_j \rangle$.
As we can see, the kernel trick is a very powerful concept, allowing to lift the input space to a potentially infinite-dimensional feature space. It is, however, important to keep in mind potential dangers of this approach.

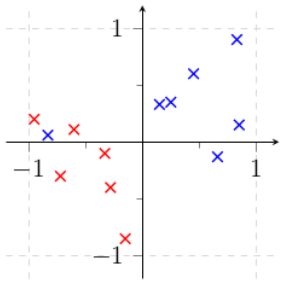


Figure 2.3: An example where one of the blue points is located in the red cluster. In this case it might be beneficial to ignore the model prediction error on that particular point in order to ensure the model generalises well

Example 2.3.1
Consider the example training data in fig. 2.3. One of the blue data points is clearly located in the red class cluster. With the kernel trick, we might be able to find a higher-dimensional feature space where all the red and blue points are linearly separable. That, however, might reduce the model's generalisation power, i.e. the model may overfit to the data. A better forward might actually be to misclassify this training example (x_i, x_j) *in order to keep the decision boundary simple and dimensionality low. We shall now discuss how the optimisation problem [2.4] can be modified to facilitate this.*

2.3.2 Allowing for misclassifications

The classifier described above maximises the margin between the classes of data points. This approach is sometimes overly optimistic as real-world data is not always linearly separable. In order to have a model that performs well on the training examples we would need to employ a complex kernel function which would likely result in the model overfitting to the training data. This line of thinking led to the introduction of the so-called *soft-margin* classifiers, which are essentially allowed to make an error. We modify the primal optimisation problem defined in eq. (2.2) to introduce the slack variables ξ_i

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \boldsymbol{w}^\top \boldsymbol{w} + C \sum_{i=1}^{N} \xi_i^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, ..., N \end{aligned} \tag{2.5}$$

The regularisation parameter C is usually chosen from a wide range of values and tuned for the problem at hand by looking at the prediction results.

2.3.3 Support vector regression

Recall that a regression task generalises a classification task by requiring the model to estimate a continuous-valued function instead of returning an output from a finite

set. Generalisation of the SVM classifier to the regression problem is achieved by introducing an ϵ-wide tube around the estimated function. The goal is to find the optimal parameters such that the estimated values lie at most ϵ distance away from the actual y_i observations. In other words, when e.g. predicting the price of an asset we would like to make sure the model forecast is not more than ϵ away from the actual price.
Instead of modelling the relationship between X and y via $f(x)$ considered in eq. (2.1), we optimise $g(x)$

$$g(x) = w^\top x + b$$

which leads to the following optimisation problem

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \boldsymbol{w}^\top \boldsymbol{w} \\ \text{s.t.} \quad & y_i - (\boldsymbol{w}^\top \boldsymbol{x}_i + b) \leq \epsilon \\ & (\boldsymbol{w}^\top \boldsymbol{x}_i + b) - y_i \leq \epsilon \\ & i = 1, ..., N \end{aligned} \tag{2.6}$$

Similar to the classification case above, we can introduce the slack variables to improve the generalising power of the model

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \boldsymbol{w}^\top \boldsymbol{w} + C \sum_{i=1}^{N} (\hat{\xi}_i^2 + \tilde{\xi}_i^2) \\ \text{s.t.} \quad & y_i - (\boldsymbol{w}^\top \boldsymbol{x}_i + b) \leq \epsilon + \hat{\xi}_i \\ & (\boldsymbol{w}^\top \boldsymbol{x}_i + b) - y_i \leq \epsilon + \tilde{\xi}_i \\ & \hat{\xi}_i, \tilde{\xi}_i \geq 0, i = 1, ..., N \end{aligned} \tag{2.7}$$

A detailed discussion of this problem, as well as associated derivations, can be found in [CS00].

2.3.4 Computational cost

The optimisation problems [2.5] and [2.7] are usually solved with the sequential minimal optimisation [Zhi+08] – a method which achieves the asymptotic time complexity of $\mathcal{O}(n^3)$, meaning that the model scales poorly with the number of training examples. However, for linear SVM optimisation problem, algorithms have been developed [Joa06] to achieve a much more favourable complexity of $\mathcal{O}(n)$. In what follows we explore methods that maintain the generalisation capabilities of a kernelised SVM, but nonetheless avoid using kernels explicitly to keep the computational cost manageable.

2.4 Kernel approximation methods

Kernel SVMs are not the only models that suffer from scalability issues. In fact, any kernelised machine learning method would be affected as it usually involves computing the $N \times N$ kernel matrix introduced in definition 2.2.5 and inverting it which, done directly, takes $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ memory. Different approaches have been proposed to alleviate this issue, such as sketching [Wan+15], low-rank matrix approximations [Fu14] and random Fourier features [RR07]. In this thesis, we shall focus on the latter approach.

2.4.1 Gaussian kernel approximation

One solution to the scalability problem appeared in the seminal work of Rahimi et. al. [RR07] which proposes an approximation scheme for the Gaussian kernel function [2.2.3] instead of calculating it exactly. With a randomised map $z : \mathbb{R}^D \mapsto \mathbb{R}^d$ the authors transform the input feature set $X \in \mathbb{R}^D$ to a lower-dimensional space $\mathbb{R}^d$ and replace the kernel evaluation in eq. (2.4) with its approximation $k(x, y) \approxeq z(x)^\top z(y)$. After the transformation, they are able to use a linear solver [Joa06], and reduce the computational complexity of the model to $\mathcal{O}(nd^2)$ time and $\mathcal{O}(nd)$ space, where $d \ll n$.
The theory behind random Fourier features builds on the following classical theorem characterizing positive definite kernel functions.

Theorem 2.4.1 (Bochner [RR07])
A continuous shift-invariant kernel $k(x, y) = k(x - y)$, $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is positive definite if and only if it is the Fourier transform of a non-negative measure. That is

$$k(x - y) = \int_{\mathbb{R}^d} \mathrm{e}^{i\omega^\top (x-y)} \mu(d\omega) = \mathbb{E}_\omega[\mathrm{e}^{i\omega^\top (x-y)}] \tag{2.8}$$

where $\mu(\omega)$ is a non-negative finite measure.

To clarify how this theorem [2.4.1] can be used in practice, we apply it to the ubiquitous Gaussian kernel as an example.

Example 2.4.1 (Application of theorem 2.4.1 to the Gaussian kernel)
Let $h(\cdot) : x \mapsto \mathrm{e}^{i\omega^\top x}$, $\omega \sim N_d(0, 1)$ and $\Delta = x - y$, then

$$\begin{aligned}
\mathbb{E}_\omega[h(x)h(y)^*] = \mathbb{E}_\omega[\mathrm{e}^{i\omega^\top x}\mathrm{e}^{-i\omega^\top y}] = \mathbb{E}_\omega[\mathrm{e}^{i\omega^\top \Delta}] &= \int_{\mathbb{R}^d} p(\omega)\mathrm{e}^{i\omega^\top \Delta} d\omega \\
&= \frac{1}{(2\pi)^{\frac{d}{2}}} \int_{\mathbb{R}^d} \mathrm{e}^{-\frac{1}{2}\omega^\top \omega}\mathrm{e}^{i\omega^\top \Delta} d\omega \\
&= \frac{1}{(2\pi)^{\frac{d}{2}}} \int_{\mathbb{R}^d} \mathrm{e}^{-\frac{1}{2}\left(\omega^\top \omega - 2i\omega^\top \Delta - \Delta^\top \Delta\right) - \frac{1}{2}\Delta^\top \Delta} d\omega \\
&= \frac{1}{(2\pi)^{\frac{d}{2}}} \mathrm{e}^{-\frac{1}{2}\Delta^\top \Delta} \int_{\mathbb{R}^d} \mathrm{e}^{-\frac{1}{2}(\omega - i\Delta)^\top (\omega - i\Delta)} d\omega \\
&= \mathrm{e}^{-\frac{1}{2}\Delta^\top \Delta} = \mathrm{e}^{-\frac{1}{2}(x-y)^\top (x-y)} = \mathrm{e}^{-\frac{||x-y||_2^2}{\sqrt{\sigma}^2}} = k(x - y)
\end{aligned}$$

where $h(x)^*$ is a complex conjugate of $h(x)$. This simple derivation uses the definition of the multivariate standard normal pdf and shows us how to approximate the Gaussian kernel function [2.2.3] for $\sigma = \sqrt{2}$.

In order to replace the complex exponential parts in the approximation, we introduce the following

$$\begin{aligned} z_\omega(x) &= \sqrt{2} cos(\omega^\top x + b) \\ b &\sim U(0, 2\pi) \end{aligned} \tag{2.9}$$

Now we can finally define the random map $z(x) : \mathbb{R}^D \mapsto \mathbb{R}^d$ by setting

$$z(x) = \begin{bmatrix} \frac{1}{\sqrt{d}} z_{\omega_1(x)} \\ \frac{1}{\sqrt{d}} z_{\omega_2(x)} \\ \vdots \\ \frac{1}{\sqrt{d}} z_{\omega_d(x)} \end{bmatrix}$$

such that

$$\begin{aligned} z(x)^\top z(x) = \frac{1}{d} \sum_1^d z_\omega(x) z_\omega(y) &= \frac{1}{d} \sum_1^d 2 cos(\omega_i^\top x + b_i) cos(\omega_i^\top y + b_i) \\ &= \frac{1}{d} \sum_1^d cos(\omega_i^\top (x - y)) \\ &\approx \mathbb{E}_\omega[cos(\omega^\top (x - y))] = \mathbb{E}_\omega[\mathrm{e}^{i\omega^\top x} \mathrm{e}^{-i\omega^\top y}] = k(x, y) \end{aligned} \tag{2.10}$$

Example 2.4.2 (Alternative embedding)
[RR07] provide an alternative feature mapping $\hat{z}$

$$\hat{z}_{\omega_i}(x) = \begin{bmatrix} cos(\omega_i^\top x) \\ sin(\omega_i^\top x) \end{bmatrix} \tag{2.11}$$

The derivation of the Gaussian kernel using this approximation can be found in the appendix eq. (A.1). *It has been shown [SS15] that the map $\hat{z}$ has a superior L_2 convergence to the true kernel function as well as uniformly lower variance than the mapping introduced in eq.* (2.10).

Example 2.4.3 (Laplace kernel)
An approximation of the Laplace kernel

$$k(x, x') = \mathrm{e}^{-\frac{||x - x'||_1}{\sigma}} \tag{2.12}$$

can be derived similarly, using the $w \sim Cauchy(\sigma)$ distribution with scale parameter σ by using the theorem 2.4.1 [Sch21].

[RR07], which introduced random Fourier features, received the NeurIPS Test-of-Time award in 2017 and the method has been recognised as one of the most popular ways to alleviate the performance bottleneck of kernel methods, making it possible to compare SVMs and Gaussian Processes with Deep Neural Networks (DNN) using the same data benchmarks. In the Gaussian kernel example above a data-independent sampling approach is used which naturally applies to streaming data. At the same time, multiple extensions of the approach have been proposed which use the distribution of data, for example leverage score sampling [Li+21], re-weighted random features [RR08]. [Liu+21] provides a detailed survey of the current state of research in the area.
To summarise, we shown that it is possible to replace the evaluation of a shift-invariant kernel with the feature map approximated using Monte Carlo simulation. In turn, this can be used to avoid large computational complexity which arises when solving the optimisation problems [2.5] and [2.7]. To reiterate, the ideas introduced here are useful beyond just SVMs, as other kernel methods also tend to suffer from the computational complexity induced by storing and inverting an $n \times n$ Gram matrix.

2.5 Path signature and its applications

2.5.1 Definition of the signature

Let's define a continous path to be a function $Y : [a, b] \mapsto \mathbb{R}^d$, where we set $a = 0, b = 1$ for simplicity. Consider now an integral of a continuous path $Y : [0, 1] \mapsto \mathbb{R}^d$ against a path of bounded variation[2] $X : [0, 1] \mapsto \mathbb{R}^d$. The integral is well-defined on any $[s, t] \subset [0, 1]$ and denoted by

$$\int_s^t Y_u dX_u = \begin{pmatrix} \int_s^t Y_u^1 dX_u^1 \\ \vdots \\ \int_s^t Y_u^d dX_u^d \end{pmatrix} \in \mathbb{R}^d \tag{2.13}$$

where $(X^1, ..., X^d)$ and $(Y^1, ..., Y^d)$ are the components of d-dimensional paths. Furthermore, if X is continuously differentiable, then eq. (2.13) is the standard Riemann integral which can be written as

$$\int_c^d Y_u dX_u = \int_c^d Y_u \dot{X}_u du$$

where $\dot{X}_u = \frac{dX}{du}$
Let's consider a sequence of indices $I = \{ i_1, ..., i_k \} \subset \{ 1, ..., d \}$ which determine the integration over a simplex

$$\Delta^k_{[a,b]} = \{ (t_1, ..., t_k) : a \leq t_1 \leq ... \leq t_k \leq b \}$$

[2] A continuous path X as defined above is said to have a finite *1-variation* if $||x||_{1-var} := sup_{\Pi \in \Delta[s,t]} \sum_{k=0}^{n-1} ||x(t_{k+1}) - x(t_k)||_1 < \infty$, where $\Delta[s, t]$ is the set of partitions of the interval. The space of continuous paths with bounded variation is denoted by $C^{1-var}([s, t], \mathbb{R}^d)$.

Then the integral of X with respect to itself can be written as

$$\int_{a\leq t_1\leq ...\leq t_k\leq b} dX_{t_1}^{i_1}...dX_{t_k}^{i_k} = \int_{a\leq t_1\leq ...\leq t_k\leq b} \dot{X}_{t_1}^{i_1}...\dot{X}_{t_k}^{i_k}dt_1...dt_k =: S(X)_{a,b}^k$$
$$k = 0, 1, 2, ...$$
$$i_1, ..., i_k \in \{ 1, ..., d \} \tag{2.14}$$

The expression above defines the *signature of level k*.

Remark 1
By convention, $S(X)_{a,b}^0 = 1 \in \mathbb{R}$*, whereas* $S(X)_{a,b}^1 = X_b - X_a \in \mathbb{R}^d$ *is the increment of path* X *over the interval* $[a, b]$*.* $S(X)_{a,b}^2$ *is a* $d \times d$ *matrix with elements*

$$\left(\int_{a\leq t_1\leq t_2\leq b} dX_{t_1}^i dX_{t_2}^j \right); i, j \in \{ 1, ..., d \}$$

In general, $S(X)_{a,b}^k$ *is a* k*-tensor with* d^k *entries*

Definition 2.5.1
The path signature of X *on the interval* $[a, b]$ *is the infinite sequence of tensors*

$$\begin{aligned} S(X)_{[a,b]} &= \left(1, S(X)_{[a,b]}^1, ..., S(X)_{[a,b]}^k, ...\right) \\ &= \left(1, \int_{a\leq t_1\leq b} dX_{t_1}, ..., \int_{a\leq t_1\leq ...\leq t_k\leq b} dX_{t_1} \otimes ... \otimes dX_{t_k}, ...\right) \end{aligned}$$

where $\otimes$ denotes tensor product.
One important fact that allows for efficient computation of the object in definition 2.5.1 is the rapid decay of the individual terms of the infinite series.

Theorem 2.5.1 (Factorial decay)
The signature terms of the series in definition 2.5.1 decay factorially, that is

$$\left\| \int_{a\leq t_1\leq ...\leq t_k\leq b} dX_{t_1} \otimes ... \otimes dX_{t_k} \right\| \leq \frac{L(X)^k}{k!}$$

Where $L(X)$ *denotes the length of the path* X *over the interval* $[a, b]$

Proof By induction, see [Lyo14]

Due to this fact it is reasonable to consider a truncated version of the signature where the series is terminated at a level $N \in \mathbb{N}$.

$$S^N(X)_{[a,b]} = \left(1, \int_{a\leq t_1\leq b} dX_{t_1}, ..., \int_{a\leq t_1\leq ...\leq t_N\leq b} dX_{t_1} \otimes ... \otimes dX_{t_N}\right) \in T^N((\mathbb{R}^d))^3 \tag{2.15}$$

We provide here two fundamental theorems which demonstrate the properties of path signatures useful for practical computations. We omit the proofs for brevity as they can be found in [LCL07].

[3] Path signature is an element of tensor algebra $T^N((\mathbb{R}^d)) = \bigoplus^N (\mathbb{R}^d)^{\otimes k}$

Theorem 2.5.2 (Chen [LCL07], Theorem 2.9)
Let $X : [0, a] \mapsto \mathbb{R}^d$ and $X : [a, b] \mapsto \mathbb{R}^d$ be two continuous paths, $X, Y \in C^{1-var}$, and let $X * Y$ be their concatenation, then

$$S(X * Y) = S(X) \otimes S(Y)$$

Theorem 2.5.3 (Signature solves an ODE [LCL07], Lemma 2.10)
The signature satisfies the following integral equation

$$S(X)_{[a,b]} = \mathbf{1} + \int_{s=a}^{b} S(X)_{[a,s]} \otimes dX_s = \mathbf{1} + \int_{s=a}^{b} S(X)_{[a,s]} \otimes \dot{X}_s ds$$

In particular, Chen's relation [2.5.2] allows for efficient practical computation of the signature of a path $X : [0, T] \mapsto \mathbb{R}^d$ as the tensor product of signatures of its pieces that comprise the linear interpolation between descrete time points $X_{t_i} = (t_i, x_i)$.

Remark 2 (Applications of truncated signature kernel in data science)
The object described in eq. (2.15) *can already be thought of as a feature transformation of the original path-valued data. For example, [Kid+19] and [LLN16] present machine learning models which use truncated signature as part of the feature set construction. However, truncating the signature at a certain level is somewhat arbitrary and leads to immediate loss of information which motivates further advancements in the use of path signatures in machine learning that we describe in the next section.*

2.5.2 The signature kernel

In section 2.2.3 we saw examples of kernel functions for different types of input data. We could define a kernel on the space of paths by computing the inner product between path signatures.

Definition 2.5.2 (Euclidean signature kernel)
The Euclidean signature kernel $k_{sig} : C^{1-var}([a, b], \mathbb{R}^d) \times C^{1-var}([a, b], \mathbb{R}^d) \mapsto \mathbb{R}$ is defined as the inner product of the signatures of the two paths $X \in C^{1-var}([a, b], \mathbb{R}^d)$ and $Y \in C^{1-var}([c, d], \mathbb{R}^d)$.

$$k_{sig}(X, Y) := \langle S(X)_{[a,b]}, S(Y)_{[c,d]} \rangle$$

where we let

$$\langle S(X)_{[a,b]}, S(Y)_{[c,d]} \rangle = 1 + \sum_{k=1}^{\infty} \langle S(X)^k_{[a,b]}, S(Y)^k_{[c,d]} \rangle$$

One practical way of using the kernel would be to compute the truncated signatures of the input paths X, Y [KO16] as defined in eq. (2.15) and calculate their inner product.

$$k_{sig}^{N}(X,Y) := \langle S^N(X)_{[a,b]}.S^N(Y)_{[c,d]} \rangle \tag{2.16}$$

Although the use of truncated signature is justified by the theorem 2.5.1, the method leads to the loss of information as pointed out in remark 2. In [Sal+21] the authors propose a method to compute untruncated signature kernel defined in 2.5.2.

Theorem 2.5.4 (The signature kernel PDE)
Let $X : [a,b] \mapsto \mathbb{R}^d$, $Y : [c,d] \mapsto \mathbb{R}^d$ be two continuously differentiable paths. Then $k_{sig}(X,Y) = u(a,d)$, and $u : [a,b] \times [c,d] \mapsto \mathbb{R}$ solves the following linear hyperbolic PDE

$$\frac{\partial^2 u}{\partial s \partial t} = u\langle \dot{X}_s, \dot{Y}_t \rangle, \text{ with } u(a,\cdot) = u(\cdot,c) = 1. \tag{2.17}$$

Proof Please see Appendix A.4

The solution of this PDE can then be approximated with a generic PDE solver, more details are provided in section 3.3.

Remark 3
The Euclidean kernel defined in 2.5.2 restricts all levels of the signature to have the same contribution to the final result. To allow for more flexibility an extension to this idea has been proposed in [CLX21]. For example, it might be useful to re-weight the terms in the sum by scaling the path X by $0 \neq \theta = \mathrm{e}^{\alpha} \in \mathbb{R}$ to obtain

$$k_{sig}(\theta X, Y) = 1 + \sum_{k=1}^{\infty} \theta^k \langle S(X)_{[a,b]}^k, S(Y)_{[c,d]}^k \rangle$$

The example in remark 3 demonstrates a rescaled inner product of the form below with $\phi(k) = 1, \forall k$

$$\langle \cdot, \cdot \rangle_{\theta\phi} = \sum_{k=0}^{\infty} \theta^k \phi(k) \langle \cdot, \cdot \rangle,$$

where $\theta \neq 0$ and $\phi : \mathbb{N}^+ \mapsto \mathbb{C}$, such that $\theta\phi : \mathbb{N}^+ \mapsto \mathbb{C}$ is a pointwise product

$$(\theta\phi)(n) = \theta^n \phi(n)$$

It is necessary to impose a summability condition on $\phi(n)$

Remark 4 (Summability condition)
The function $\phi : \mathbb{N}^+ \mapsto \mathbb{C}$ is such that

$$\sum_{k \in \mathbb{N}} \frac{C^k \phi(k)}{k!^2} < \infty \text{ for every } C > 0$$

Definition 2.5.3 (The general signature kernel)
Given the two paths $X \in C^{1-var}([a,b],\mathbb{R}^d)$ and $Y \in C^{1-var}([c,d],\mathbb{R}^d)$, the function ϕ which satisfies the condition in remark 4 and $\theta \neq 0$, the general signature kernel is defined as

$$k_{sig}^{\theta\phi} := \langle S(X)_{[a,b]}.S(Y)_{[c,d]}\rangle_{\theta\phi} = 1 + \sum_{k=1}^{\infty} \theta^k \phi(k) \langle S(X)_{[a,b]}.S(Y)_{[c,d]}\rangle \tag{2.18}$$

Remark 5
When $\theta = 1$ and $\phi \equiv 1$ we recover the Euclidean signature kernel from definition 2.5.2

Remark 6 (Scaled signature equation)
Let $\theta \in \mathbb{R}$ and $\phi : \mathbb{N}^+ \mapsto \mathbb{C}$ which satisfies the condition in remark 4, then

$$k_{sig}^{\theta\phi}(X,Y) = k_{sig}^{\phi}(\theta X, Y) = k_{sig}^{\phi}(X, \theta Y)$$

moreover, if $\phi \equiv 1$ then $k_{sig}^{\theta\phi}(X,Y) = k_{sig}^{\theta}(X,Y)$ satisfies

$$k_{sig}^{\theta}(X,Y) = 1 + \theta \int_{u=a}^{b} \int_{v=c}^{d} k_{sig}^{\theta}(s,t) \langle \dot{X}, \dot{Y} \rangle du dv$$

To summarise, we progressed from the limited information that can be obtained by truncating the signature of a path and using it as part of a feature transform to obtain a much richer representation of the two paths via the general signature transform defined in [2.5.3]. Equipped with this powerful mathematical framework, we proceed by discussing various forms that $\phi(k)$ can take, how a numerical solution of [2.5.4] can be found and, finally, design a model to perform learning and inference on real-world data.

Chapter 3

Model design

In this chapter we combine the instruments considered in chapter 2 to present a composite model which can be used to solve supervised learning problems with high-dimensional financial time series.

3.1 Considered general signature kernels

We begin by reviewing the reweighing scheme which motivated the introduction of the general signature kernel in remark 3.

Remark 7 (Exponentially weighted kernel)
Let $\alpha \in \mathbb{R} < 1$ and $\theta = \mathrm{e}^{\alpha}$, then $k^{\theta}_{sig}(X, Y)$ defines an exponentially weighted kernel and satisfies the integral equation in remark 6, which can be solved by rescaling the solution obtained from eq. (3.4)

The scaling does not necessarily have to be deterministic, consider a random variable π with some probability measure μ on $\mathbb{R}$ which has finite moments $\mathbb{E}[\pi^k]$. Then if $\phi(k)$ satisfies the condition in remark 4 and

$$\phi(k) = \mathbb{E}[\pi^k] = \int \pi^k d\mu(\pi), \forall k \geq 0$$

Then the general signature kernel $k^{\phi}_{sig}(s,t)$ of continuous bounded variation paths X and Y is well-defined and

$$\begin{aligned} k^{\phi}_{sig}(s,t) &= \langle S(X)_{[a,b]}.S(Y)_{[c,d]} \rangle_{\phi} \\ &= \sum_{k=0}^{\infty} \phi(k) \langle S(X)_{[a,b]}.S(Y)_{[c,d]} \rangle = \sum_{k=0}^{\infty} \int \pi^k d\mu(\pi) \langle S(X)^k_{[a,b]}.S(Y)^k_{[c,d]} \rangle \\ &= \int \sum_{k=0}^{\infty} \pi^k \langle S(X)^k_{[a,b]}.S(Y)^k_{[c,d]} \rangle d\mu(\pi) = \int k^{\pi}_{sig}(s,t) d\mu(\pi) \\ &= \mathbb{E}\left[k^{\pi}_{sig}(s,t)\right] \end{aligned} \tag{3.1}$$

If the random variable π has a closed-form density, the kernel described in eq. (3.1) can be computed by using a numerical integration scheme, such as the Gaussian

Quadrature with weights w_i by running the PDE solver m times where m depends on the desired precision

$$\int k^{\pi}_{sig}(s,t)d\mu(\pi) = \sum_{i=1}^{m} w_i k^{\pi_i}_{sig}(s,t) \tag{3.2}$$

Remark 8
The same steps could in fact be applied to any integral transform of the form

$$\phi(u) = \int_C r(u,z)d\mu(z), \text{ where } r(u,z) = g(z)^{\alpha u} \in \mathbb{C}, \text{ for some } \alpha \in \mathbb{R}$$

Which include Fourier-, Laplace- and Mellin-Stieljies transforms

The examples below demonstrate the application of this method to moment generating functions of various random variables.

Example 3.1.1 (Uniform RV)
Suppose $\pi \sim U(0,1)$ is a standard uniform random variable, such that its moments match $\phi(k) = \frac{1}{k+1}$. The general signature kernel under this scaling is then

$$\begin{aligned} k^{\phi}_{sig}(s,t) = \sum_{k=0}^{\infty} \frac{1}{k+1} \langle S(X)^k_{[a,b]}.S(Y)^k_{[c,d]} \rangle &= \int_0^1 \sum_{k=0}^{\infty} \pi^k \langle S(X)^k_{[a,b]}.S(Y)^k_{[c,d]} \rangle d\mu(\pi) \\ &= \mathbb{E}_{\pi}\left[k^{\pi}_{sig}(s,t)\right] \end{aligned}$$

Example 3.1.2 (Rayleigh RV)
Let $\pi \sim Exp(1)$, then for the random variable $\pi^{1/2}$ the moments would coincide with

$$\mathbb{E}[\pi^{k/2}] = \int_0^{\infty} x^{k/2} \mathrm{e}^{-x} dx = \Gamma\left(\frac{k}{2}+1\right) = \left(\frac{k}{2}\right)! = \phi(k)$$

The general signature kernel under this scaling is then

$$k^{\phi}_{sig}(s,t) = \mathbb{E}_{\pi}[k^{\pi^{1/2}}_{sig}(s,t)]$$

Note: $\pi^{1/2}$ is Rayleigh distributed

For other examples of general signature kernels see Appendix B.1.
These examples demonstrate how we can design weighting schemes for the general signature kernel using various random variable distributions. A simple implication of that is we can learn the optimal distribution (or weighting scheme) as part of the model training process. We consider the choice of general signature kernel form to be the first building block of our model. We use *Const*, *Uniform* and *Rayleigh* labels for the kernels defined in [7], [3.1.1], [3.1.2] respectively.

3.2 Gaussian Quadrature

To calculate general signature kernels we need to be able to approximate the value of the integral mentioned in eq. (3.2), or, in general, the integral of the following form

$$\int_a^b f(x)w(x)dx = \sum_{i=0}^{\infty} w_i f(\xi_i) \approxeq \sum_{i=0}^{m} w_i f(\xi_i)$$

Where the integrated function f will typically be the solution of the scaled signature equation: $f(\theta) = k^{\theta}_{sig}(X, Y)$, and $w(x) \in L^1(a, b)$ is a positive weight function.
The standard way of approximating this integral is via the Gaussian Quadrature method [SM03]. Although the Quadrature Rule is commonly defined for the interval $[-1, 1]$, we can convert the limits of integration to any interval $[a.b]$ in the following way

$$\begin{aligned} \int_a^b f(x)dx &= \int_{-1}^{1} f\left(\frac{b-a}{2}\xi + \frac{a+b}{2}\right)\frac{dx}{d\xi}d\xi \\ &\text{where } \frac{dx}{d\xi} = \frac{b-a}{2} \\ &\text{so that } \int_a^b f(x)dx \approxeq \frac{b-a}{2}\sum_{i=0}^{m} w_i f\left(\frac{b-a}{2}\xi_i + \frac{a+b}{2}\right) \end{aligned} \tag{3.3}$$

Example 3.2.1 (Uniform RV)
Change of limits can be applied to compute the general signature kernel $k^{\phi}_{sig}(s, t)$ with $\phi(k) = \frac{1}{1+k}$. Applying the transformation eq. (3.3) *to the results in example 3.1.1 we obtain*

$$k^{\phi}_{sig}(s, t) = \int_0^1 k^{x}_{sig}(s, t)dx \approxeq \frac{1}{2}\sum_{i=0}^{m} w_i f\left(\frac{1}{2}(\xi_i + 1)\right)$$

where $f(\theta) = k^{\theta}_{sig}(X, Y)$ and w_i, ξ_i are the weights and abscissae for the standard Quadrature Rule.

Example 3.2.2 (Rayleigh RV)
Going back to the example where $\phi(k) = \left(\frac{k}{2}\right)!$. As discussed in example 3.1.2, we need to evaluate the integral of the following form

$$\begin{aligned} k^{\phi}_{sig}(s, t) = \mathbb{E}_{\pi}[k^{\pi^{1/2}}_{sig}(s, t)] &= \int_0^{\infty} f(x)g_{\pi^{1/2}}(x)dx \\ &= 2\int_0^{\infty} f(x)x\mathrm{e}^{-x^2}dx \approxeq 2\sum_{i=0}^{m} w_i f(\xi_i) \end{aligned}$$

where $g_{\pi^{1/2}}(x) = 2x\mathrm{e}^{-x^2}$ is the probability distribution function of Rayleigh r.v. $\pi^{1/2}$, $f(x)$ is the evaluation of scaled signature kernel $f(\theta) = k^{\theta}_{sig}(X, Y)$ and the weight

function in the definition of Quadrature Rule becomes $w(x) = xe^{-x^2}$. *The standard Quadrature Rule cannot be applied in this case as the integral limits and the function to be integrated are different, however this case is treated in great detail in [Shi81], where explicit tables for Quadrature weights* w_i *and abscissae* ξ_i *for the integral of the form* $\int_0^\infty f(x)xe^{-x^2}dx$ *can be found.*

Quadrature rule for *Beta* general signature kernel can also be derived but is omitted here for brevity, see Appendix B.2.

3.3 Numerical solution

The solution to the kernel signature PDE introduced in theorem 2.5.4 can be obtained by applying a simple finite difference approximation [Sal+21] which we discuss here.
We let

$$\mathcal{D}_I = \{\, a = u_0 < u_1 < \ldots < u_{m-1} < u_m = b \,\}$$

and

$$\mathcal{D}_J = \{\, c = v_0 < v_1 < \ldots < v_{m-1} < v_m = d \,\}$$

be the partitions of the time interval of the input paths X and Y. Using the *forward difference* approximation on the grid $\mathcal{P}_1 = \mathcal{D}_I \times \mathcal{D}_J$ we discretise the differential operator

$$\begin{aligned}\frac{\partial}{\partial s}\left(\frac{\partial u(s,t)}{\partial t}\right) &= \frac{\partial}{\partial s}\left(\frac{u(s,t+\Delta t)-u(s,t)}{\Delta t}\right) = \frac{\partial}{\partial s}\left(\frac{u(s,t+\Delta t)}{\Delta t}\right) - \frac{\partial}{\partial s}\left(\frac{u(s,t)}{\Delta t}\right) \\ &= \frac{u(s+\Delta s,t+\Delta t)-u(s,t+\Delta t)}{\Delta s \Delta t} - \frac{u(s+\Delta s,t)-u(s,t)}{\Delta s \Delta t}\end{aligned}$$

Recall that the signature PDE states [2.5.4]

$$\frac{\partial^2 k_{sig}}{\partial s \partial t} = k_{sig}(s,t)\langle \dot{X}, \dot{Y}\rangle$$

By using the discretisation above for the $\frac{\partial^2 k_{sig}}{\partial s \partial t}$ and taking unit step size $\Delta t = \Delta s = 1$ we obtain

$$u(s+\Delta s,t+\Delta t) - u(s,t+\Delta t) - u(s+\Delta s,t) + u(s,t) = u(s,t)\langle \dot{X}_s, \dot{Y}_t\rangle$$

so that

$$u(s+\Delta s,t+\Delta t) = u(s,t+\Delta t) + u(s+\Delta s,t) - u(s,t)\left(1 - \langle \dot{X}_s, \dot{Y}_t\rangle\right)$$

leading to the following approximation

$$\begin{aligned}\hat{k}(u_{i+1},v_{j+1}) = \hat{k}(u_i,v_{j+1}) &+ \hat{k}(u_{i+1},v_j) \\ &- \hat{k}(u_i,v_j)\left(1 - \langle X_{u_{i+1}} - X_{u_i}, Y_{v_{j+1}} - Y_{v_j}\rangle\right)\end{aligned} \tag{3.4}$$

Further improvements to the accuracy of the scheme, such as *central difference* approximation and *dyadic refinement* of the grid are discussed in [Sal+21], as well as plots showing the numerical error distribution.

Remark 9 (Computational complexity of the signature kernel)
Note that the computational complexity of calculating the signature kernel is $\mathcal{O}(Dl^2)$ for D-dimensional paths X of length l for the original signature kernel and is further exacerbated in the general signature case by the integral approximation via the Quadrature Rule. This is prohibitively expensive for long time series and motivates the research done as part of the following chapter.

3.4 Static kernel approximation and the model

The signature kernels discussed in [KO16], [Sal+21], [CLX21] provide a way to compute the kernel of the paths of values in $\mathbb{R}^D$ lifted from the input space to the feature space by the static kernel, such as linear [2.2.1], Gaussian [2.2.3] or Laplace [2.12]. In this section we explore if it is possible to apply the kernel approximation ideas from section 2.4 to the signature learning framework and whether we can gain any performance benefits by doing this.
Recall the definition of the signature kernel in theorem 2.5.4. The inner product $\langle \dot{X}, \dot{Y} \rangle$ is the static kernel on paths. In the case of the Gaussian or Laplace kernel, we can approximate this function using the random Fourier features mapping

$$\frac{\partial^2 k_{sig}}{\partial s \partial t} = k_{sig}(s,t)\langle \dot{X}, \dot{Y} \rangle \approxeq k_{sig}(s,t) h(\dot{X})^\top h(\dot{Y})$$

where $h(\cdot)$ is one of the mappings defoned in section 2.4.1

$$z(x) = \begin{bmatrix} \frac{1}{\sqrt{d}} z_{\omega_1(x)} \\ \vdots \\ \frac{1}{\sqrt{d}} z_{\omega_d(x)} \end{bmatrix}, \quad \hat{z}_{\omega_i}(x) = \begin{bmatrix} cos(\omega_i^\top x) \\ sin(\omega_i^\top x) \end{bmatrix}$$

that are applied to embed the input from $\mathbb{R}^D$ into a lower-dimensional $\mathbb{R}^d$, $d \ll D$ in order to reduce the complexity of the signature kernel computation to $\mathcal{O}(dl^2)$.

Remark 10
Note, the computational complexity of the general signature kernel combined with the RFF kernel approximation is still dominated by the quadratic term in the length of the time series.The proposed solution demonstrates the possibility to compute the general signature of the random feature embeddings as well as reduce the dimensionality of the data as much as $\sqrt{D}$ while maintaining a comparable model performance as presented in the empirical results section.

We now outline the final algorithm for the proposed model, which combines the ideas developed in parts [3.1], [3.2], [3.3] of this chapter. The schematic diagram of the algorithm can be found in fig. 3.1.

- The first step is data pre-processing, apart from splitting the data into training, validation and test sets it includes the transformations detailed in table 3.1.

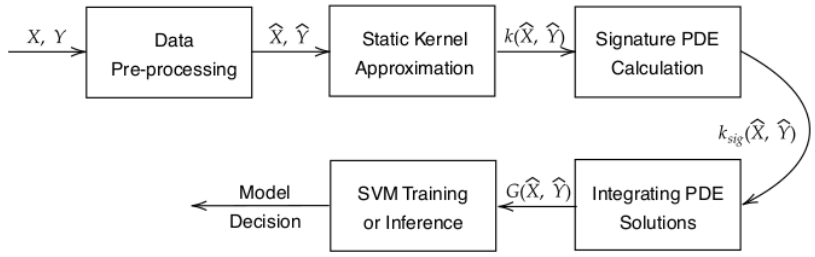


Figure 3.1: The diagram displays the model components which are used to train and test the proposed model.

Table 3.1: Hyperparameters definition of the data transformation step of the model

Name	Description	Values
`scale`	input paths scaling	$\{0.01, 0.1, 1\}$
`add_time`	adding time axis	`True`, `False`
`add_lead_lag`	adding lead-lag features (see [3.4.1] for definition)	`True`, `False`

- The pre-processed D-dimensional data is then fed to the static kernel approximation module which embeds the data into a lower d-dimensional space using a feature map. This transformation admits the hyperparameters in table 3.2.
- The general signature kernel implementation is selected from the *Const* [7], *Uniform* [3.1.1] and *Rayleigh* [3.1.2] options.
- For *Uniform* and *Rayleigh* the developed quadrature formulae are used, the number of nodes (precision) implicitly set to be 16.
- Finally, `scikit-learn` SVC/SVR implementation allows to configure the regularisation parameter $C \in \{1, 10, ..., 10^4\}$ which is used in the optimisation problem [2.5].

Note that scaling the path by γ is essentially equivalent to applying a *Const* weighting kernel with the scaling parameter $\theta = \gamma$, see [2.5.3].

Definition 3.4.1 (Lead-lag transformation)
Given a stream $(X_{t_i})^N \in \mathbb{R}^D$ *the lead transformation* $f(X_j^{lead})^{2N}$ *is a stream of* (X_{t_j}, X_{t_k}) *where* $j = 2i$ *and* $k = 2i - 1$*, whereas the lag transform will work in the opposite direction, namely* (X_{t_j}, X_{t_k}) *where* $j = 2i$ *and* $k = 2i + 1$.

Next we evaluate the ideas discussed in this chapter using the benchmark and financial data.

Table 3.2: Hyperparameters definition of the static kernel approximation step of the model

Name	Description	Values
`rff_features`	the dimensionality of the target space	$\{ \sqrt{D}, \frac{D}{4}, \frac{D}{2} \}$
`rff_metric`	the distribution of ω in $z_\omega(\cdot)$	`Laplace`, `Gaussian`
`rff_use_offset`	the flag which controls whether $z_\omega(\cdot)$ [2.9] (when set to `True`) or $\hat{z}_\omega(\cdot)$ [2.11] is used	`True`, `False`
`rbf_sigma`	Gaussian/Laplace static kernel σ parameter	$[10^{-3}, 10^{-2}, ...1, 2]$

Chapter 4

Experimental Results

In this chapter we would like to first demonstrate the advantages of the general signature framework over the original signature PDE approach in [Sal+21]. In the original signature kernel paper it was shown that the signature PDE is superior in performance to the alternative models, such as truncated signature from [KO16] and Linear, RBF [2.2.3] and GAK [Cut+07] kernels.
Our first goal is to consider the same classification problem on the benchmark UEA [Bag+18] datasets as presented in the original signature PDE paper [Sal+21] and showcase the advantages of the general signature kernel kernel in the ability to generalise. Additionally we would like to compare the best results achieved for each of the datasets with a version with the static kernel approximation that uses the random Fourier features approach.
In the second set of experiments we consider the problem of BTC price forecasting using the high-dimensional time series of factors - something that general signature kernel should be able to take advantage of - which was outlined in [Mud+20]. The authors surveyed several machine learning models for the task at hand and concluded that Long short-term memory (LSTM) model outperforms others in the majority of cases.

4.1 UEA time series classification

Problem setting

In section 2.3 we discussed the use of support vector machines for classification problems. The authors of [Sal+21] apply SVC model with various kernel functions to the problem of classifying the UEA time series. In this section we aim to demonstrate the advantages of using the general signature kernel for this classification task by running the model on the same benchmark. Some of the datasets from the original collection have been omitted due to their extremely large size which causes memory and processing power issues on a regular PC. However, we believe that any results obtained from running on the selected datasets are generic and given a more powerful machine we would have been able to replicate them across other datasets.
All datasets are of the following structure

Table 4.1: Main parameters and dimensions of the datasets used for classification task

Dataset Name	Train Size, N	Test Size, $\hat{N}$	Length, M	Classes, L	Features, D
BasicMotions	40	40	100	4	6
Libras	180	180	45	15	2
NATOPS	180	180	51	6	24
ERing	30	270	65	6	4
RacketSports	151	152	30	4	6

- X - a $N \times M \times D$ tensor, where N is the total number of samples, M is the length of each sample in time steps and D is the number of features sampled at each of the time intervals.
- Y - a vector of size N, where each value represents a class from $\{1, ..., L\}$.

Datasets

The subset of considered data includes the following datasets

- BasicMotions: 3D data from accelerometer and gyroscope of a smart watch device recorded for 10 seconds, while doing one of four activities - running, walking, resting and badminton
- Libras: 45-frame recording of hand movement represented as a bi-dimensional curve, where each sample is a hand movement type in the official Brazilian sign language
- NATOPS: the 3D coordinates of 8 sensors on the body of an air traffic controller who uses hands to display various commands on an airfield
- ERing: consists of 65 4-dimensional observations of measurements of an electric field tracked by a ring prototype. Each series in the dataset corresponds to one of 6 finger shapes.
- RacketSports: the X, Y, Z coordinate of the gyroscope recorded by a smart watch device while playing squash and badminton. 4 classes represent various strokes in each of the sports

For the exact breakdown between train/test set as well as dataset dimensions please refer to the table 4.1.

Hyperparameter tuning

In this experiment we aim to match the setup described in the original signature paper [Sal+21] and only add the hyperparameters related to the general signature kernel. The train and test split is recommended by the UEA benchmark methodology as shown in table 4.1, and we tune the hyperparameters using a 5-fold cross-validation grid search on the training set.

The hyperparameters include: `add_time`, `add_lead_lag`, `scale` the input time series by a factor [3.1], as well as the standard kernel parameters, C and RBF σ taken in the range $\{10^0, 10^1, ..., 10^4\}$ and $\{10^{-3}, 10^{-2}, ...1, 2\}$ respectively. Since the general signature kernel uses the same static kernel, the hyperparameters are the same with the addition of a scaling constant e^i for the *Const* kernel, where $i \in \pm\{5^{-3}, 5^{-2}, ..., 5^0, 5^1\}$.

Methodology

To assess the performance of the model we chose the accuracy score metric, which computes the fraction of correct predictions on the test set displayed in percentage units.

$$accuracy(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1} \mathbb{1}_{\hat{y}_i = y_i} \tag{4.1}$$

The metric was chosen because the datasets we worked with are balanced in terms of the distribution of classes.

Results and discussion

The results of the experiment described above can be found in the table 4.2. It is clear that the general signature kernel provides superior performance to the original signature *Sig-PDE* and the truncated signature kernel *Sig(n)* in all considered scenarios. We do. however, see varying performance amongst the general signature kernels alone, but there is consistency in the performance of the exponentially weighted scaling[7] and the uniform RV[3.1.1] scaling schemes. Notably, the training accuracy score of the exponentially weighted constant scaling is consistently higher than the alternative models, however this does not seem to be the case for the test set, which susggests that the model might be overfitting to the dataset and additional regularisation is required. We believe that prior to choosing the scaling parameter of *Const* kernel it is worth doing exploratory data analysis to see the scaling effect on individual terms of the signature kernel.

4.2 UEA classification - static kernel approximation

Problem setting

In this section we would like to demonstrate the dimensionality reduction achieved by using the approximated static kernel as described in fig. 3.1 and compare the performance of the proposed model with the results from section 4.1. Since we need at least a few dimensions before we can reduce them, the *Libras* dataset is excluded as it is only 2-dimensional (max 4-dimensional with lead-lag transformation).

Table 4.2: Accuracy scores (%) computed using eq. (4.1) of the different models considered. The performance metrics of Sig(n) and Sig-PDE models are taken from the [Sal+21], whereas the rest are computed using the implementation of the models considered.

Kernel	Sig(n)	Sig-PDE	Const	Uniform	Rayleigh
NATOPS	N/A	90.6	**95**	93.3	93.3
Libras	N/A	74.4	**76.1**	73.3	74.4
RacketSports	N/A	91.4	92.7	**93.3**	92.0
ERing	N/A	93.3	**96.7**	93.3	93.3
BasicMotions	N/A	97.5	**100**	97.5	**100**

Training accuracy

Kernel	Sig(n)	Sig-PDE	Const	Uniform	Rayleigh
NATOPS	88.3	93.3	87.8	88.3	**95.5**
Libras	81.7	81.7	87.7	**88.3**	87.7
RacketSports	80.2	84.9	82.2	**86.2**	82.9
ERing	84.1	92.2	**92.6**	92.2	91.5
BasicMotions	97.5	**100**	**100**	**100**	**100**

Testing accuracy

Hyperparameter tuning

In addition to the hyperparameters of the first experiment we can now also tune the kernel approximation hyperparameters described in table 3.2.

Methodology

We follow the same methodology as in section 4.1 to evaluate the performance of the model and also record the ratio between the input space and the approximated space dimensionality (including dimensions added via lead-lag and time transformations).

Results and discussion

The results of this experiment can be found in table 4.3. We first observe the model score varies across different datasets, in fact, in most of the cases the performance is reduced within 15% of the original accuracy of the full kernel, however for the *ERing* data the drop in performance is around 30%. It is worth to point out that the proportion of the training set is prescribed to be much smaller for *ERing* than for any other dataset, see table 4.1. Because the precision of random features approximation is bounded by the number of samples [RR07], this particular dataset suffers most.
On the other hand, for the remaining datasets we see a dramatic decrease in the data dimensionality from D to d of up to 80% with only about a 10% decrease in performance. The proposed general signature kernel still seems to outperform the original *Sig-PDE* kernel, but it is difficult to choose a particular implementation because, for

Table 4.3: Accuracy scores (%) for the model with static kernel approximation, as well as dimensionality reduction comparison of accuracy scores with the general signature kernel without static kernel approximation in table 4.2

Dataset	Kernel					
	Sig-PDE			Const		
	Accuracy, %	Reduction, %		Accuracy, %	Reduction, %	
		Dim.	Acc.		Dim.	Acc.
NATOPS	83.9	76.0	10.0	78.3	79.2	10.8
RacketSports	**77.0**	57.1	9.3	75.7	57.1	7.9
ERing	66.7	60.0	27.6	61.4	80.0	33.7
BasicMotions	95.0	78.6	5.0	**97.5**	78.6	2.5

Results for the original signature kernel and *Const* general signature

Dataset	Kernel					
	Uniform			Rayleigh		
	Accuracy, %	Reduction, %		Accuracy, %	Reduction, %	
		Dim.	Acc.		Dim.	Acc.
NATOPS	82.8	76.0	6.2	**85.0**	76.0	11.0
RacketSports	72.4	57.1	16.0	74.3	57.1	10.4
ERing	**67.9**	60.0	26.4	64.0	80.0	30.0
BasicMotions	95.0	57.1	5.0	**97.5**	57.1	2.5

Results for the *Uniform* and *Rayleigh* general signature

example, *Rayleigh* model displays a better accuracy score, but also suffered most from the static kernel approximation.
To conclude, the results seem promising, as the approximation looks to achieve comparable results to the baseline while reducing the dimensionality of the input data set, however additional empirical evidence is required to demonstrate that it is data-independent and universally applicable. In a later section 4.4 we test the proposed model on the regression problem as well.

4.3 Bitcoin price prediction with a range of technical indicators

Problem setting

The goal of this section is to compare the predictive performance of the model suggested in chapter 3 with the recently published research surveying multiple approaches for the problem of forecasting the times series of Bitcoin prices using various technical indicators [Mud+20]. In their work, Mudassir et. al. argue that machine learning models are better equipped to deal with the non-stationarity and high volatility of the price time series than traditional time series models.

To demonstrate this they collect a large dataset of predictor variables X and estimate the value of a target variable y at a future time point s, i.e. $\hat{y}[t+s] = f(X[t], X[t-1], ..., X[t-n])$, where t is the current time, s is the forecast horizon taken to be 1 (end of day (EOD)), 7, 30 and 90 days and n is the lookback window chosen to be 30 days. They conduct a few regression and classification experiments and compare the performance of the artificial neural network (ANN) and its stacked version (SANN), support vector machine (SVM) and long short-term memory (LSTM) network on these tasks. According to their results, LSTM and SANN are the two top performers, which agrees with what other authors have shown previously [MRC18], so our aim is to test if the general signature kernel provides any advantages to the traditional machine learning setup.

Dataset

The authors of [Mud+20] use `https://bitinfocharts.com/`, the publicly available source of cryptocurrency data to collect more than 700 features based on price, technical indicators and information about the transactions in the blockchain. They proceed by applying the random forest machine learning model to determine feature importance and filter the dataset to only leave the subset of features with relatively high importance scores, low cross-correlation metric and no multi-colinearity. The feature selection step is performed for each of the s future forecast horizons and the authors find that the subsets of important features for each of the horizons are not necessarily the same.

The missing data points have been imputed via linear interpolation or by choosing the most frequently occurring value for categorical data. The values have been normalised using the MinMax transformation to keep them between 0 and 1, importantly, this was done without leaking any future information by fitting the MinMax transformer on the training window only. The data was further split into three time periods: Interval I from April to September 2013, Interval II from August 2015 to March 2017 and Interval III from March 2017 to November 2018. Each of the intervals is posing a distinctive challenging task for the forecasting model e.g., sharp reversion of the market in 2018 or a strong trend in the beginning of 2017, and the remaining dataset was split 80/20 into train and test parts. We follow the same methodology and describe the final feature set used in table 4.4.

Hyperparameter tuning

In this chapter we used almost the same set of hyperparameters as in the previous one. The only difference is the lead lag hyperparameter which was always set to *False* to avoid leaking the future data during model training. In addition to that, there was no time series scaling except for the MinMax scaling as described in the Dataset section.

Table 4.4: Description of the features used for each of the forecasting horizons: 1 (EOD), 7, 30 and 90 days. Refer to [Mud+20] for details.

Features	Description	EOD	7	30	90
median_transaction_fee30trxUSD	30-day and 7-day triple moving exponential smoothing of the median transaction fee of BTC	*		*	*
median_transaction_fee7trxUSD		*			*
price90emaUSD	90-day exponentially moving average price	*			*
size90trx	90-day triple moving exponential smoothing block size	*	*	*	*
transactions	The number of sent and received Bitcoin payments	*			*
price30wmaUSD	The 3-day, 7-day, 30-day	*		*	
price3wmaUSD	and 90-day weighted	*	*	*	
price7wmaUSD	moving average price of	*	*		
price90wmaUSD	Bitcoin in USD			*	*
median_transaction_fee7USD	The 7-day median transaction fee	*	*		
mining_profitability	The profitability in USD/day for 1 terahash per second		*	*	*
sentinusd90emaUSD	The 90-day exponentially moving average of the total Bitcoin sent daily		*	*	
top100cap	The ratio between the top-100 Bitcoin holders and the rest		*		*
difficulty30rsi	The 30-day relative strength indicator and 90-day momentum of the average mining difficulty	*	*	*	*
difficulty90mom				*	*
hashrate90var	The 90-day variance of the computational power of the Bitcoin network			*	*
transactionvalueUSD	The daily median transaction value of Bitcoin in USD		*	*	*
median_transaction_feeUSD	Median transaction fee received by the miners who verify the transaction			*	*

Methodology

In order to assess the model performance we use the mean absolute percentage error (MAPE) metric calculated with eq. (4.2) and represented in percentage terms. To match with the authors of the paper [Mud+20] we also include mean absolute error (MAE)[4.3] and mean squared error (MSE)[4.4]. A model with lower errors is desirable, but ultimately it is up to the researcher to choose the appropriate metric for the task at hand. We would like to pay particular attention to the MSE to see how large the outliers in the model forecasts are compared to the actual data, as well as MAPE due to the fact it is easy to compare percentage errors across models and experiments.

Remark 11 (Regression model assessment metrics)

$$MAPE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} \frac{|y_i - \hat{y}_i|}{|y_i|} \tag{4.2}$$

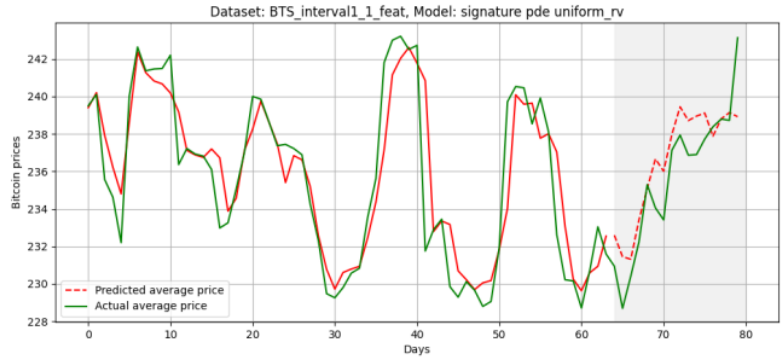
$$MAE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - \hat{y}_i| \tag{4.3}$$

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2 \tag{4.4}$$

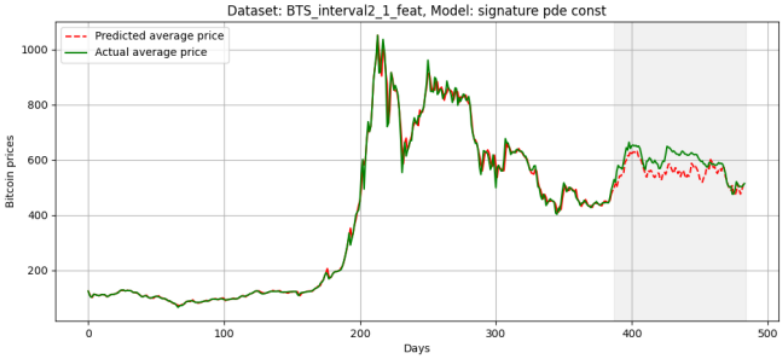
Results and discussion

In the first experiment we compare the EOD price forecasts for intervals I, II and III, in this case the *Benchmark* model is the LSTM network as it performed best in the paper (see table 5 in [Mud+20]). We observe immediately (see table 4.5) that the Interval II was challenging for any of the Signature kernel models, the errors are much higher across the board than the benchmark error. During testing on this dataset we also noticed several instances of the learning algorithm failing to converge, which could be the reason for poor performance. Our intuition is the dataset imputation/construction added noise to the data, so numerical methods such as PDE solver and integration were imprecise. As for the rest of the data, it is clear that the Signature kernel is better or comparable in terms of generalisation power. For the Interval I, the *Unifrom* general signature seems to be performing the best, whereas Interval III is split between *Sig-PDE* and *Rayleigh*. We can also observe that MSE errors for Signature kernel models are slightly higher which could mean the model does not generalise well to the presence of outliers in the data. For comparison of actual vs predicted data see fig. 4.1, where the models with lowest errors are included, otherwise refer to section C.1.1 in Appendix for other plots.
For the next experiment the authors fixed the time interval to III and considered the comparison of forecasts of the price of Bitcoin for 7, 30 and 90 days ahead. The models are assessed using the same set of metrics as in the first experiment, however this time the *Benchmark* model is chosen to be SANN as it clearly outperforms the others in the paper (see table 6 in [Mud+20]). The results are presented in the

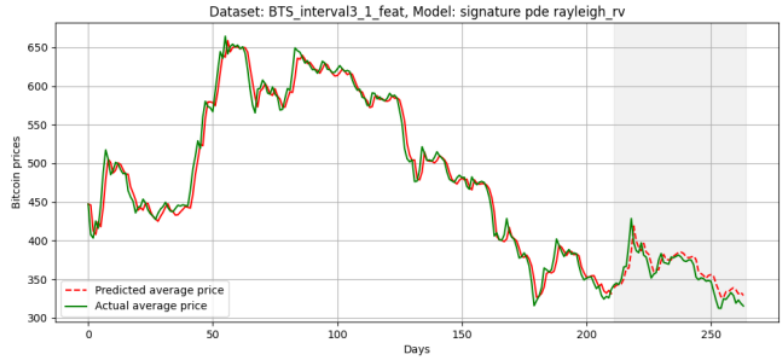
table 4.6. Looking at the 7-day forecast horizon it is difficult to choose which model



(a) Price prediction for EOD in the Interval I using the *Uniform* model



(b) Price prediction for EOD in the Interval II using the *Const* model



(c) Price prediction for EOD in the Interval III using the *Rayleigh* model

Figure 4.1: Predicted vs. actual Bitcoin price for time intervals I (a), II (b) and III (c)

Table 4.5: Forecasting the end of day price using the time intervals I, II and III described in the paper [Mud+20]. Results for the benchmark model are taken from table 5 of the paper for the LSTM network which seemed to perform best amongst the models the authors considered.

Metric	Intervals	Benchmark	Sig-PDE	Const	Uniform	Rayleigh
MAE	I	2.20	1.49	1.43	**1.33**	**1.53**
	II	**6.55**	10.54	10.98	17.30	17.41
	III	62.90	**5.54**	6.12	12.96	6.11
MSE	I	3.01	4.16	3.67	**2.32**	4.39
	II	**10.55**	199.62	105.68	215.00	197.54
	III	135.76	**50.16**	58.95	133.86	67.35
MAPE, %	I	0.93	0.63	0.69	**0.56**	0.621
	II	1.98	1.88	**1.87**	5.77	5.78
	III	3.61	2.44	2.47	3.65	**2.32**

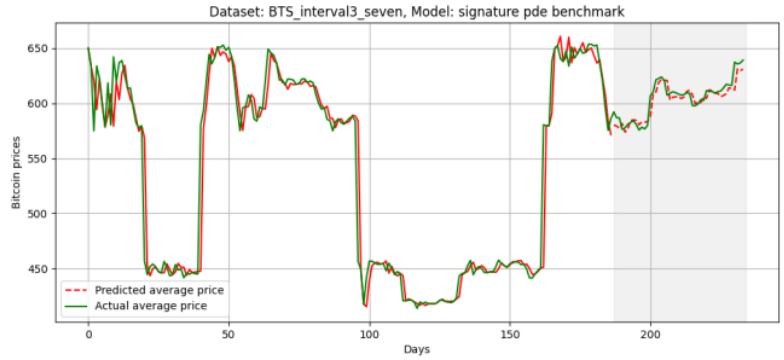
Table 4.6: Forecasting the Bitcoin price 7, 30 and 90 days ahead using the Interval III and respective features described in table 4.4. Results for the benchmark model are taken from table 6 of the paper for the SANN model.

Metric	Horizon	Benchmark	Sig-PDE	Const	Uniform	Rayleigh
MAE	7	16.32	**13.69**	13.71	16.55	33.56
	30	77.12	81.46	82.47	**68.96**	83.02
	90	72.23	63.26	64.20	65.24	**62.51**
MSE	7	36.33	34.12	34.14	**27.17**	125.79
	30	156.30	173.84	174.85	**156.10**	190.61
	90	**140.00**	220.27	212.77	216.07	205.12
MAPE, %	7	**2.88**	2.95	2.96	3.47	4.97
	30	3.45	3.06	3.17	**2.57**	3.08
	90	4.10	3.63	3.65	3.90	**3.53**

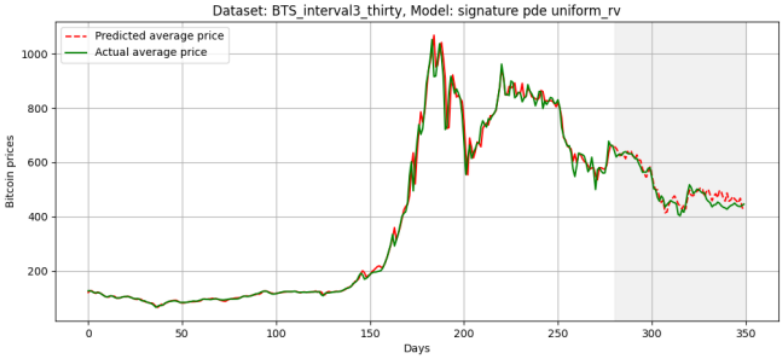
performed the best, since the *Benchmark* is better in terms of MAPE, but *Uniform* has a lower MSE. When we consider the 30-day and 90-day forecasts it becomes clear that *Uniform* and *Rayleigh* are the top models for these horizons respectively. This experiment shows the ability of the general signature kernel to generalise to high-dimensional datasets with non-trivial relationship between the variables. Interestingly, the peformance of *Sig-PDE* and *Const* models is very similar for this test, certainly more similar than for the same time window in table 4.5. It looks like bits of relevant information about the interaction between time series are getting lost due to the constant (although learnable in case of *Const*) factorial decay term at each of the signature levels. For reference, we present the plots detailing the actual vs. predicted price in fig. 4.2 for the top performing models for each of the time horizons. Plots for all models can be found later in section C.1.2 of the Appendix.

The next two experiments use the same data setup to address the problem of forecasting the direction of the future movement of the price. In particular, the price data is first converted to classes in $\{0, 1\}$, where 0 by comparing the values on sub-

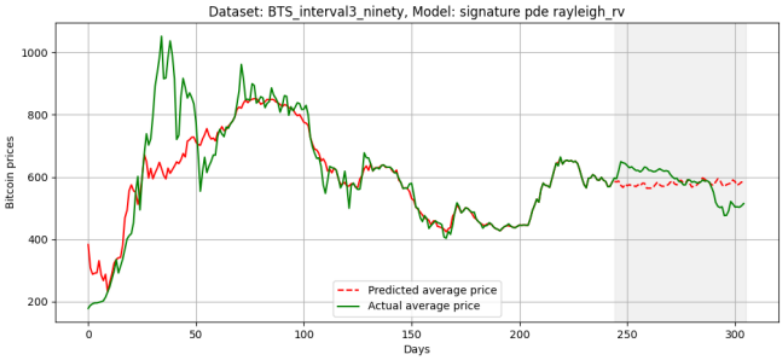
sequent days. Since the dataset is essentially the same as in the regression example,



(a) Price prediction for 7 days ahead using the *Sig-PDE* model



(b) Price prediction for 30 days ahead using the *Uniform* model



(c) Price prediction for 90 days ahead using the *Rayleigh* model

Figure 4.2: Predicted vs. actual Bitcoin price for 7 (a), 30 (b) and 90 (c) days ahead

Actual price	Predicted price	
	Decrease	Increase
Decrease	TN	FP
Increase	FN	TP

Table 4.7: Confusion matrix for the Bitcoin price direction forecasting in experiments 3 and 4. The cells TN, FN are the number of True and False negatives obtained as part of the evaluation - where the model predicted the price decrease, and the actual price decreased and increased respectively. In the second column we have numbers of False and True positives - these are when the model predicted price increase, and the real price decreased and increased respectively. Confusion matrix is a useful tool for classification model assessment which is we use it for.

we choose to limit the trained models to only $Uniform$ to save some computing time. To assess the performance of the classifier we used the accuracy score, F-score and area under the curve (AUC), which are described in remark 12.

Remark 12 (Classification model assessment metrics)
The accuracy metric presented in 4.1 is commonly used for classification problems, however if the dataset is imbalanced (in finance this could be a trending time series) different metrics such as F-score and AUC might be more suitable [Pat+20].
For F-score we need to introduce precision and recall which are calculated using the confusion matrix described in table 4.7. On the same dataset we prefer the model with a higher F-score.

$$\textit{Precision} = \frac{TP}{TP+FP}, \textit{Recall} = \frac{TP}{TP+FN}, \textit{Specificity} = \frac{TN}{TN+FP}$$

$$\textit{F-score} = \frac{2 \times \textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}} \tag{4.5}$$

AUC score is the area under the receiver operating characteristic (ROC) curve plotted with Recall along the y-axis and Specificity along the x-axis. The intuitive explanation of the AUC is the probability that the model ranks a random positive example higher than the random negative one [Man10], it ranges from 0 to 1 and a value of 0.5 means the model cannot discriminate between the two classes.

The results of this experiment are presented in the table 4.8 for multiple intervals setup and table 4.9 for multiple horizons setup. To see where the model ranks compared to the alternative machine learning solutions, we include the results from the tables 9 and 10 of [Mud+20] respectively and highlight our model in blue. In the first experiment the *Uniform* model ranks fairly well on the intervals I and II, whereas the scores for interval III suggest the model failed to generalise. Looking at the regression problem results for the interval III [4.5] we see a similar issue with *Uniform* model, which seems consistent. Surprisingly, the training accuracy of the model for interval III is close to 89% suggesting that the model performed fairly well on the training set. One explanation of such discrepancy would be that there is an

Metric	Interval	ANN	SVM	SANN	LSTM	Uniform
Accuracy, %	I	57	55	**60**	54	56
	II	56	62	65	53	**72**
	III	53	56	**60**	54	40
F-score	I	**0.72**	0.67	0.71	0.63	0.68
	II	0.69	0.74	0.75	0.58	**0.83**
	III	0.61	0.53	0.60	**0.66**	0.56
AUC	I	0.51	0.51	**0.56**	0.52	0.46
	II	0.50	0.55	0.59	0.53	**0.62**
	III	0.53	0.56	**0.60**	0.54	0.50

Table 4.8: This table compares the *Uniform* general signature model with the benchmark results from [Mud+20]. In the test we predict EOD price for the next day on different date intervals.

Metric	Horizon	ANN	SVM	SANN	LSTM	Uniform
Accuracy, %	7	51	62	53	55	**70**
	30	52	52	62	52	**71**
	90	62	54	60	**64**	40
F-score	7	**0.65**	0.58	0.38	**0.65**	0.43
	30	0.68	0.68	**0.70**	0.68	0.65
	90	0.56	0.66	0.68	**0.71**	0.23
AUC	7	0.53	0.60	0.51	0.56	**0.62**
	30	0.50	0.50	0.61	0.50	**0.70**
	90	0.59	0.57	0.62	**0.66**	0.28

Table 4.9: This table compares the *Uniform* general signature model with the benchmark results from [Mud+20]. In the test we predict prices for 7, 30 and 90 day horizons.

imbalance between the training/test data which also seems to affect the benchmark models from the paper. In the multiple horizons experiment the model seems to perform quite well on shorted days ahead intervals, but poorly handles the 90 days ahead window. This suggests that the model captures regularities on a short horizon better than the long one. It is also worth mentioning that we do not completely agree with the problem statement as explained in the remark 13.

Remark 13
The goal of the model in the experiments 3 and 4 is to forecast the direction of price movement at some point in the future using a certain interval of training data. Although valid, it is not a very practical as the information from the model does not say anything about the magnitude of the movement and it is not clear how small fluctuations around an arbitrary value should be handled. A more pragmatic approach would solve this issue by e.g., defining 3 classes of price fluctuations: up, down and unchanged, or aim to forecast even more specific cases such as trend direction [AK21], strength [LZR20] or its speed [Che+22].

	Sig-PDE	Const	Uniform	Rayleigh
MAPE, %	12.24	12.60	11.76	**10.06**

Table 4.10: MAPE scores on BTC data for Interval III and various general signature kernels. 6 random Fourier features have been used.

4.4 Bitcoin price prediction with a range of technical indicators - static kernel approximation

Problem setting

In the final set of experiments we include the random Fourier features mapping in our model setup and consider again the problem of forecasting Bitcoin price for one day ahead using the methodology discussed in section 4.3. To assess the model we run the experiments on the Interval III for different general signature kernels with the number of random features set to 6 (approx. 50% dimensionality reduction). MAPE scores and qualitative plots are collected.
Moreover, we include a test where we vary the number of random projection components and collect the MAPE scores. We have chosen $\{ 3, 6, 9, 19 \}$ random features for this test to assess the performance across the whole range. Note, that the last option is larger than the number of dimensions we started with, which is 14. It has been added to show that the random projections can also be used to map input data into higher dimensions.

Dataset

The dataset used in this chapter is the EOD feature set from table 4.4 which has been transformed using the same steps as in section 4.3.

Methodology

In order to verify the model performance we use MAPE metric defined in eq. (4.2) and check the plots of forecasted vs. actual prices.

Results and discussion

Results for each of the signature kernel alternatives are presented in the table 4.10, Consistent with the experiment in section 4.3, *Rayleigh* general signature kernel shows the smallest relative error. In general, we see that model accuracy dropped, but it is still reasonable, which is impressive considering that the dimensionality of data reduced by more than 50%.
Qualitative plots comparing the forecast and actual prices for this experiment have been put in Appendix, please refer to fig. C.4.
In the second experiment we see [4.11] that *Rayleigh* model with 9 random feature projections outperformed the rest in terms of MAPE metric. Looking at the plots [C.6] of the price forecast we observe high variance in the forecasts made by 3

Fourier features	3	6	9	18
MAPE, %	18.89	18.25	**8.4**	11.58

Table 4.11: MAPE scores on BTC data for Interval III and various general signature kernels. 6 random Fourier features have been used.

and 6-feature models. This is, perhaps, because the sample size is too small for model calibration for such a low resolution. On the other hand, we see that the model with 19 feature vectors did not show significant outperformance confirming the dimensionality of the original dataset is enough to train a good model.

Chapter 5

Conclusion

In this thesis we provide the first implementation of generals signature kernel ideas presented in [CLX21]. We discuss the possible model design choices made during the practical implementation of the model and, finally, provide a systematic comparison of the proposed model against the signature kernel introduced in [Sal+21] as well as other machine learning models mentioned in [Mud+20].
Additionally, we explore the idea of static kernel approximation via the random Fourier features [RR07]. We use the same benchmark data to test the model with kernel approximation and outline the benefits and downsides of this approach.
Overall, the general signature kernel (2.5.3) represents a more expressive model that its original signature counterpart (2.5.4), as confirmed by the empirical results. In section 4.1 we show that various general signature implementations uniformly outperform the original signature and truncated signature (2.15) versions. Furthermore, in the regression task discussed in section 4.3, the general signature kernel model performs better or comparable with the best proposed machine learning model on that particular dataset. These experiments confirm the general signature framework is universal and flexible, and can be used in many different applications of multivariate time series modelling.
The main drawback of the general signature kernel approach is the prohibitive computational complexity for large time series, the PDE solution alone runs in $\mathcal{O}(Dl^2m)$ operations, where D is the dimensionality of the space, l is the length of time series and m is the number of nodes in the numerical integration (precision). In section 3.4 we propose a solution based on the idea of random feature mappings that reduces the cost of storage and compute by as much as $\sqrt{D}$ once the mapping has been applied. We validate the feasibility of this approach in section 4.2 on the classification task and section 4.4 on the regression task for multivariate time series. Although, unsurprisingly, the predictive performance of the model is somewhat reduced (while remaining adequate), this comes with a large dimensionality reduction of up to 80% in some cases.
These observations lead to a handful of promising avenues for future research. Firstly, the general signature kernel relies on sequential numerical integration of PDE solutions. Any improvement achieved by exploiting the relationship between these solutions would lead to improvements in the computational cost as this part is the backbone of the whole model.

Moreover, we have considered only a few possibilities for general signature weighting schemes, but as we mentioned in remark 8, the framework which was proposed by [CLX21] is very generic and allows for other integral transformations. Thus, any research in finding the relationship between the form of a general signature kernel and specific properties of the data would be very insightful.
Furthermore, we find the fact that signature kernel performs well on the random feature mappings insightful. There are quite a few different kernel approximation methodologies, such as the ones based on sketching [Wan+15], [CCF04], random feature maps [LIS10], [RR07], [Liu+21] or sparse variational inference [TO20]. Thus it is reasonable to think that further research in applying these techniques would lead to the improvement in computational efficiency for the powerful concept of general signature kernels.

Bibliography

[AG19] Robert Adcock and Nikola Gradojevic. "Non-fundamental, non-parametric Bitcoin forecasting". In: *Physica A: Statistical Mechanics and its Applications* 531 (Oct. 2019), p. 121727. ISSN: 03784371. DOI: 10.1016/j.physa.2019.121727. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378437119309859 (visited on 09/04/2022).

[AK21] Adesola Adegboye and Michael Kampouridis. "Machine learning classification and regression models for predicting directional changes trend reversal in FX markets". In: *Expert Systems with Applications* 173 (July 2021), p. 114645. ISSN: 09574174. DOI: 10.1016/j.eswa.2021.114645. URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417421000865 (visited on 08/31/2022).

[ASR88] Milton Abramowitz, Irene A. Stegun, and Robert H. Romer. "*Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*". In: *American Journal of Physics* 56.10 (Oct. 1988), pp. 958–958. ISSN: 0002-9505, 1943-2909. DOI: 10.1119/1.15378. URL: http://aapt.scitation.org/doi/10.1119/1.15378 (visited on 09/04/2022).

[Bag+18] Anthony Bagnall et al. *The UEA multivariate time series classification archive, 2018*. Oct. 31, 2018. arXiv: 1811.00075[cs,stat]. URL: http://arxiv.org/abs/1811.00075 (visited on 09/03/2022).

[CCF04] Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding frequent items in data streams". In: *Theoretical Computer Science* 312.1 (Jan. 2004), pp. 3–15. ISSN: 03043975. DOI: 10.1016/S0304-3975(03)00400-6. URL: https://linkinghub.elsevier.com/retrieve/pii/S0304397503004006 (visited on 09/06/2022).

[CF19] Alessandra Cretarola and Gianna Figà-Talamanca. "Modeling Bitcoin Price and Bubbles". In: *Blockchain and Cryptocurrencies*. Ed. by Asma Salman and Muthanna G. Abdul Razzaq. IntechOpen, Aug. 28, 2019. ISBN: 978-1-78923-913-3 978-1-83881-208-9. DOI: 10.5772/intechopen.79386. URL: https://www.intechopen.com/books/blockchain-and-cryptocurrencies/modeling-bitcoin-price-and-bubbles (visited on 09/04/2022).

[Che+22] Eddie Cheng et al. "Trending Fast and Slow". In: *The Journal of Portfolio Management* 48.3 (Jan. 31, 2022), pp. 103–116. ISSN: 0095-4918, 2168-8656. DOI: 10.3905/jpm.2021.1.312. URL: http://jpm.pm-

research.com/lookup/doi/10.3905/jpm.2021.1.312 (visited on 08/31/2022).

[CLN12] Run Cao, Xun Liang, and Zhihao Ni. "Stock Price Forecasting with Support Vector Machines Based on Web Financial Information Sentiment Analysis". In: *Advanced Data Mining and Applications*. Ed. by Shuigeng Zhou, Songmao Zhang, and George Karypis. Red. by David Hutchison et al. Vol. 7713. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 527–538. ISBN: 978-3-642-35526-4 978-3-642-35527-1. DOI: 10.1007/978-3-642-35527-1_44. URL: http://link.springer.com/10.1007/978-3-642-35527-1_44 (visited on 09/04/2022).

[CLX21] Thomas Cass, Terry Lyons, and Xingcheng Xu. "General Signature Kernels". In: (2021). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2107.00447. URL: https://arxiv.org/abs/2107.00447 (visited on 06/12/2022).

[CS00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. 1st ed. Cambridge University Press, Mar. 23, 2000. ISBN: 978-0-521-78019-3 978-0-511-80138-9. DOI: 10.1017/CBO9780511801389. URL: https://www.cambridge.org/core/product/identifier/9780511801389/type/book (visited on 09/03/2022).

[CS08] Andreas Christmann and Ingo Steinwart. *Support Vector Machines*. Information Science and Statistics. ISSN: 1613-9011. 2008. DOI: 10.1007/978-0-387-77242-4. URL: http://link.springer.com/10.1007/978-0-387-77242-4.

[Cut+07] Marco Cuturi et al. "A kernel for time series based on global alignments". In: *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*. Apr. 2007, pp. II–413–II–416. DOI: 10.1109/ICASSP.2007.366260. arXiv: cs/0610033. URL: http://arxiv.org/abs/cs/0610033 (visited on 07/11/2022).

[Det+18] Andrew L. Detzel et al. "Bitcoin: Predictability and Profitability via Technical Analysis". In: *SSRN Electronic Journal* (2018). ISSN: 1556-5068. DOI: 10.2139/ssrn.3115846. URL: https://www.ssrn.com/abstract=3115846 (visited on 09/04/2022).

[Det+21] Andrew Detzel et al. "Learning and predictability via technical analysis: Evidence from bitcoin and stocks with hard-to-value fundamentals". In: *Financial Management* 50.1 (Mar. 2021), pp. 107–137. ISSN: 0046-3892, 1755-053X. DOI: 10.1111/fima.12310. URL: https://onlinelibrary.wiley.com/doi/10.1111/fima.12310 (visited on 09/04/2022).

[Fu14] Zhouyu Fu. “Optimal Landmark Selection for Nyström Approximation”. In: *Neural Information Processing*. Ed. by Chu Kiong Loo et al. Vol. 8835. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 311–318. ISBN: 978-3-319-12639-5 978-3-319-12640-1. DOI: 10.1007/978-3-319-12640-1_38. URL: http://link.springer.com/10.1007/978-3-319-12640-1_38 (visited on 09/06/2022).

[Gau83] Walter Gautschi. “How and how not to check Gaussian quadrature formulae”. In: *BIT Numerical Mathematics* 23.2 (June 1983), pp. 209–216. ISSN: 0006-3835, 1572-9125. DOI: 10.1007/BF02218441. URL: https://link.springer.com/10.1007/BF02218441 (visited on 09/04/2022).

[Gba+21] Adedeji Daniel Gbadebo et al. “BTC price volatility: Fundamentals versus information”. In: *Cogent Business & Management* 8.1 (Jan. 1, 2021), p. 1984624. ISSN: 2331-1975. DOI: 10.1080/23311975.2021.1984624. URL: https://www.tandfonline.com/doi/full/10.1080/23311975.2021.1984624 (visited on 09/04/2022).

[Gyu+14] Lajos Gergely Gyurkó et al. *Extracting information from the signature of a financial data stream*. July 15, 2014. arXiv: 1307.7244[q-fin]. URL: http://arxiv.org/abs/1307.7244 (visited on 09/04/2022).

[JL18] Huisu Jang and Jaewook Lee. “An Empirical Study on Modeling and Prediction of Bitcoin Prices With Bayesian Neural Networks Based on Blockchain Information”. In: *IEEE Access* 6 (2018), pp. 5427–5437. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2779181. URL: http://ieeexplore.ieee.org/document/8125674/ (visited on 09/04/2022).

[Joa06] Thorsten Joachims. “Training linear SVMs in linear time”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’06*. the 12th ACM SIGKDD international conference. Philadelphia, PA, USA: ACM Press, 2006, p. 217. ISBN: 978-1-59593-339-3. DOI: 10.1145/1150402.1150429. URL: http://portal.acm.org/citation.cfm?doid=1150402.1150429 (visited on 09/05/2022).

[Kid+19] Patrick Kidger et al. “Deep Signature Transforms”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/d2cdf047a6674cef251d56544a3cf029-Paper.pdf.

[KO16] Franz J. Király and Harald Oberhauser. *Kernels for sequentially ordered data*. Jan. 29, 2016. arXiv: 1601.08169[cs,math,stat]. URL: http://arxiv.org/abs/1601.08169 (visited on 07/02/2022).

[LCL07] Terry J. Lyons, Michael Caruana, and Thierry Lévy. *Differential Equations Driven by Rough Paths: École d’Été de Probabilités de Saint-Flour XXXIV - 2004*. Red. by J.-M. Morel Cachan, F. Takens Groningen, and B. Teissier Paris. Vol. 1908. Lecture Notes in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. ISBN: 978-3-540-71284-8

978-3-540-71285-5. DOI: 10.1007/978-3-540-71285-5. URL: http://link.springer.com/10.1007/978-3-540-71285-5 (visited on 06/19/2022).

[Li+21] Zhu Li et al. *Towards A Unified Analysis of Random Fourier Features*. Feb. 4, 2021. arXiv: 1806.09178[cs,stat]. URL: http://arxiv.org/abs/1806.09178 (visited on 09/03/2022).

[LIS10] Fuxin Li, Catalin Ionescu, and Cristian Sminchisescu. "Random Fourier Approximations for Skewed Multiplicative Histogram Kernels". In: *Pattern Recognition*. Ed. by Michael Goesele et al. Red. by David Hutchison et al. Vol. 6376. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 262–271. ISBN: 978-3-642-15985-5 978-3-642-15986-2. DOI: 10.1007/978-3-642-15986-2_27. URL: http://link.springer.com/10.1007/978-3-642-15986-2_27 (visited on 09/06/2022).

[Liu+21] Fanghui Liu et al. *Random Features for Kernel Approximation: A Survey on Algorithms, Theory, and Beyond*. July 11, 2021. arXiv: 2004.11154[cs,stat]. URL: http://arxiv.org/abs/2004.11154 (visited on 08/31/2022).

[LLN16] Daniel Levin, Terry Lyons, and Hao Ni. *Learning from the past, predicting the statistics for the future, learning an evolving system*. Mar. 22, 2016. arXiv: 1309.0260[q-fin]. URL: http://arxiv.org/abs/1309.0260 (visited on 09/04/2022).

[Lyo14] Terry Lyons. *Rough paths, Signatures and the modelling of functions on streams*. Number: arXiv:1405.4537. May 18, 2014. arXiv: 1405.4537[math,q-fin,stat]. URL: http://arxiv.org/abs/1405.4537 (visited on 05/23/2022).

[LZR20] Bryan Lim, Stefan Zohren, and Stephen Roberts. *Enhancing Time Series Momentum Strategies Using Deep Neural Networks*. Sept. 27, 2020. arXiv: 1904.04912[cs,q-fin,stat]. URL: http://arxiv.org/abs/1904.04912 (visited on 08/31/2022).

[Man10] Jayawant N. Mandrekar. "Receiver Operating Characteristic Curve in Diagnostic Test Assessment". In: 5.9 (Sept. 2010), pp. 1315–1316. ISSN: 15560864. DOI: 10.1097/JTO.0b013e3181ec173d. URL: https://linkinghub.elsevier.com/retrieve/pii/S1556086415306043 (visited on 08/31/2022).

[Mar+11] Eric Martin et al. "String kernel". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2011, pp. 929–929. ISBN: 978-0-387-30768-8 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_790. URL: http://link.springer.com/10.1007/978-0-387-30164-8_790 (visited on 07/11/2022).

[MR99] A.R. Meenakshi and C. Rajian. “On a product of positive semidefinite matrices”. In: *Linear Algebra and its Applications* 295.1 (July 1999), pp. 3–6. ISSN: 00243795. DOI: 10.1016/S0024-3795(99)00014-2. URL: https://linkinghub.elsevier.com/retrieve/pii/S0024379599000142 (visited on 09/04/2022).

[MRC18] Sean McNally, Jason Roche, and Simon Caton. “Predicting the Price of Bitcoin Using Machine Learning”. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). Cambridge: IEEE, Mar. 2018, pp. 339–343. ISBN: 978-1-5386-4975-6. DOI: 10.1109/PDP2018.2018.00060. URL: https://ieeexplore.ieee.org/document/8374483/ (visited on 08/30/2022).

[Mud+20] Mohammed Mudassir et al. “Time-series forecasting of Bitcoin prices using high-dimensional features: a machine learning approach”. In: *Neural Computing and Applications* (July 4, 2020). ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-020-05129-6. URL: https://link.springer.com/10.1007/s00521-020-05129-6 (visited on 08/29/2022).

[Pat+20] Harshita Patel et al. “A review on classification of imbalanced data for wireless sensor networks”. In: *International Journal of Distributed Sensor Networks* 16.4 (Apr. 2020), p. 155014772091640. ISSN: 1550-1477, 1550-1477. DOI: 10.1177/1550147720916404. URL: http://journals.sagepub.com/doi/10.1177/1550147720916404 (visited on 08/31/2022).

[RR07] Ali Rahimi and Benjamin Recht. “Random Features for Large-Scale Kernel Machines”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt et al. Vol. 20. Curran Associates, Inc., 2007.

[RR08] Ali Rahimi and Benjamin Recht. “Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller et al. Vol. 21. Curran Associates, Inc., 2008. URL: https://proceedings.neurips.cc/paper/2008/file/0efe32849d230d7f53049ddc4a4b0c60-Paper.pdf.

[Sal+21] Cristopher Salvi et al. “The Signature Kernel Is the Solution of a Goursat PDE”. In: *SIAM Journal on Mathematics of Data Science* 3.3 (Jan. 2021), pp. 873–899. ISSN: 2577-0187. DOI: 10.1137/20M1366794. URL: https://epubs.siam.org/doi/10.1137/20M1366794 (visited on 06/19/2022).

[SC04] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. OCLC: 144618454. Cambridge, UK; New York: Cambridge University Press, 2004. ISBN: 978-0-511-21060-0 978-0-511-21597-1 978-0-511-21237-6 978-0-511-80968-2 978-0-511-21418-9. URL: https://doi.org/10.1017/CBO9780511809682 (visited on 07/03/2022).

[Sch21] Vincent Schellekens. “Extending the Compressive Statistical Learning Framework: Quantization, Privacy, and Beyond”. PhD thesis. UCLouvain, Aug. 6, 2021. URL: https://schellekensv.github.io/Resources/thesis.pdf.

[Shi+15] Ali Shiri et al. “Electricity price forecasting using Support Vector Machines by considering oil and natural gas price impacts”. In: *2015 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*. 2015 IEEE International Conference on Smart Energy Grid Engineering (SEGE). Oshawa, ON, Canada: IEEE, Aug. 2015, pp. 1–5. ISBN: 978-1-4673-7932-8. DOI: 10.1109/SEGE.2015.7324591. URL: http://ieeexplore.ieee.org/document/7324591/ (visited on 09/04/2022).

[Shi81] B Shizgal. “A Gaussian quadrature procedure for use in the solution of the Boltzmann equation and related problems”. In: *Journal of Computational Physics* 41.2 (June 1981), pp. 309–328. ISSN: 00219991. DOI: 10.1016/0021-9991(81)90099-1. URL: https://linkinghub.elsevier.com/retrieve/pii/0021999181900991.

[SKA21] Ndeye Fatou Sene, Mamadou Abdoulaye Konte, and Jane Aduda. “Pricing Bitcoin under Double Exponential Jump-Diffusion Model with Asymmetric Jumps Stochastic Volatility”. In: *Journal of Mathematical Finance* 11.2 (2021), pp. 313–330. ISSN: 2162-2434, 2162-2442. DOI: 10.4236/jmf.2021.112018. URL: https://www.scirp.org/journal/doi.aspx?doi=10.4236/jmf.2021.112018 (visited on 09/04/2022).

[SM03] Endre Süli and D. F. Mayers. *An introduction to numerical analysis*. OCLC: ocm50525488. Cambridge ; New York: Cambridge University Press, 2003. 433 pp. ISBN: 978-0-521-81026-5 978-0-521-00794-8.

[SS] Bernhard Scholkopf and Alexander J Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. OCLC: 904718080. Cambridge; Ipswich: MIT Press ; Ebsco Publishing [distributor, 0. ISBN: 978-0-262-25693-3. URL: http://ieeexplore.ieee.org/servlet/opac?bknumber=6267332 (visited on 07/03/2022).

[SS15] Danica J. Sutherland and Jeff Schneider. *On the Error of Random Fourier Features*. June 9, 2015. arXiv: 1506.02785[cs,stat]. URL: http://arxiv.org/abs/1506.02785 (visited on 09/03/2022).

[TO20] Csaba Toth and Harald Oberhauser. *Bayesian Learning from Sequential Data using Gaussian Processes with Signature Covariances*. July 6, 2020. arXiv: 1906.08215[cs,math,stat]. URL: http://arxiv.org/abs/1906.08215 (visited on 09/06/2022).

[USC07] Christian Ullrich, Detlef Seese, and Stephan Chalup. “Foreign Exchange Trading with Support Vector Machines”. In: *Advances in Data Analysis*. Ed. by Reinhold Decker and Hans -J. Lenz. Series Title: Studies in Classification, Data Analysis, and Knowledge Organization. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 539–546. ISBN: 978-3-540-70980-0 978-3-540-70981-7. DOI: 10.1007/978-3-540-70981-7_62.

URL: http://link.springer.com/10.1007/978-3-540-70981-7_62 (visited on 09/04/2022).

[Vis+08] S. V. N. Vishwanathan et al. *Graph Kernels*. July 1, 2008. arXiv: 0807.0093[cs]. URL: http://arxiv.org/abs/0807.0093.

[Wan+15] Yining Wang et al. *Fast and Guaranteed Tensor Decomposition via Sketching*. Oct. 20, 2015. arXiv: 1506.04448[cs,stat]. URL: http://arxiv.org/abs/1506.04448 (visited on 09/06/2022).

[Xie+06] Wen Xie et al. "A New Method for Crude Oil Price Forecasting Based on Support Vector Machines". In: *Computational Science – ICCS 2006*. Ed. by Vassil N. Alexandrov et al. Red. by David Hutchison et al. Vol. 3994. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 444–451. ISBN: 978-3-540-34385-1 978-3-540-34386-8. DOI: 10.1007/11758549_63. URL: http://link.springer.com/10.1007/11758549_63 (visited on 09/04/2022).

[Zhi+08] Zhi-Qiang Zeng et al. "Fast training Support Vector Machines using parallel sequential minimal optimization". In: *2008 3rd International Conference on Intelligent System and Knowledge Engineering*. 2008 3rd International Conference on Intelligent System and Knowledge Engineering (ISKE 2008). Xiamen, China: IEEE, Nov. 2008, pp. 997–1001. ISBN: 978-1-4244-2196-1. DOI: 10.1109/ISKE.2008.4731075. URL: http://ieeexplore.ieee.org/document/4731075/ (visited on 09/05/2022).

[ZPC11] Shuji Zhao, Frédéric Precioso, and Matthieu Cord. "Spatio-Temporal Tube data representation and Kernel design for SVM-based video object retrieval system". In: *Multimedia Tools and Applications* 55.1 (Oct. 2011), pp. 105–125. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-010-0602-3. URL: http://link.springer.com/10.1007/s11042-010-0602-3 (visited on 09/04/2022).

Appendices

Appendix A

A.1 Proof of theorem 2.2.1

$$k(x,x) = \sum_{i,j} c_i c_j G_{i,j} = \sum_{i,j} c_i c_j k(x_i, x_j) = \sum_{i,j} \langle c_i \phi(x_i), c_j \phi(x_j) \rangle_{\mathcal{H}}$$
$$= \left\langle \sum_i c_i \phi(x_i), \sum_j c_j \phi(x_j) \right\rangle_{\mathcal{H}} = \left\| \sum_i c_i \phi(x_i) \right\|_{\mathcal{H}}^2 \geq 0$$

A.2 Proof of theorem 2.2.2

Fix a finite set of data $\{ \boldsymbol{x}_1, ..., \boldsymbol{x}_N \}$ and let K_1, K_2 be the matrices obtained by restricting the kernel functions k_1, k_2 to this data set. Recall that a matrix K is positive semi-definite *iff* for any $\boldsymbol{\alpha} \in \mathbb{R}^N$, $\alpha^\top K \alpha \geq 0$, then

- (sum of kernels) $\alpha^\top (K_1 + K_2)\alpha = \alpha^\top K_1 \alpha + \alpha^\top K_2 \alpha \geq 0$, so $K_1 + K_2$ is a positive semi-definite matrix meaning that $k_1 + k_2$ is a valid kernel. For βk where $\beta > 0$ just plug β in the definition of a positive semi-definite kernel which will not change the inequality.

- The tensor product of two positive semi-definite matrices K_1 and K_2 is positive semi-definite [MR99], the result of the theorem follows trivially.

A.3 Derivation of an alternative feature mapping [2.4.2] for random Fourier features

We begin by defining the alternative map as in [RR07]

$$\hat{z}_{\omega_i}(x) = \begin{bmatrix} cos(\omega_i^\top x) \\ sin(\omega_i^\top x) \end{bmatrix} \tag{A.1}$$

We follow by drawing $\bar{d} = \frac{d}{2}$ samples from $p(\omega)$ and compute

$$\begin{aligned}
\frac{1}{\bar{d}}\sum_{1}^{\bar{d}} z_{\omega_i}(x)z_{\omega_i}(y) &= \frac{2}{\bar{d}}\sum_{1}^{\bar{d}}\left(\begin{bmatrix}cos(\omega_i^\top x)\\ sin(\omega_i^\top x)\end{bmatrix}^\top \begin{bmatrix}cos(\omega_i^\top y)\\ sin(\omega_i^\top y)\end{bmatrix}\right) \\
&= \frac{2}{\bar{d}}\sum_{1}^{\bar{d}} cos(\omega_i^\top x)cos(\omega_i^\top y) + sin(\omega_i^\top x)sin(\omega_i^\top y) \\
&= \frac{2}{d}\sum_{1}^{d/2} cos(\omega_i^\top x - \omega_i^\top y) \\
&\approxeq \mathbb{E}_\omega[cos(\omega_i^\top (x-y))] = k(x,y)
\end{aligned} \tag{A.2}$$

A.4 Derivation of the signature kernel PDE [2.5.4]

$$\begin{aligned}
k_{sig}(s,t) &= \langle S(X)_s, S(Y)_t\rangle \\
&= \langle 1 + \int_{u=a}^{b} S(X)\otimes d(X), 1 + \int_{v=c}^{d} S(Y)\otimes d(Y)\rangle \quad (2.5.3) \\
&= 1 + \int_{u=a}^{b}\int_{v=c}^{d} \langle S(X)\otimes \dot{X}, S(Y)\otimes \dot{Y}\rangle dudv \quad \text{(by differentiability)} \\
&= 1 + \int_{u=a}^{b}\int_{v=c}^{d} \langle S(X), S(Y)\rangle\langle \dot{X}, \dot{Y}\rangle dudv \\
&= 1 + \int_{u=a}^{b}\int_{v=c}^{d} k_{sig}(s,t)\langle \dot{X}, \dot{Y}\rangle dudv \quad (2.5.2)
\end{aligned}$$

By the fundamental theorem of calculus we can differentiate the above first by s and t to obtain:

$$\frac{\partial^2 k_{sig}}{\partial s \partial t} = k_{sig}(s,t)\langle \dot{X}, \dot{Y}\rangle$$

Appendix B

B.1 General signature kernel using *Beta*-distributed random variable

Remark 14 (Beta RV)
Let $\pi \sim Beta(1, m)$, the sequence of its moments is

$$\mathbb{E}[\pi^k] = \frac{B(k+1,m)}{B(1.m)} = \frac{\Gamma(k+1)\Gamma(m+1)}{\Gamma(k+m+1)} =: \phi(k)$$

The general signature kernel under the scaling $\phi(k)$ is

$$k^{\phi}_{sig}(s,t) = \mathbb{E}_{\pi}[k^{\pi}_{sig}(s,t)]$$

B.2 Quadrature rule for *Beta*-weighted general signature kernel

Remark 15 (Beta RV)
Taking $\pi \sim Beta(1, m)$ as in remark 14, we have the following integral to evaluate

$$\begin{aligned}
k^{\phi}_{sig}(s,t) = \mathbb{E}_{\pi}[k^{\pi}_{sig}(s,t)] &= \int_0^1 f(x) g_{\pi}(x) dx \\
&= \frac{1}{B(1,m)} \int_0^1 f(x)(1-x)^{m-1} dx = \frac{1}{B(1,m)} \int_0^1 f(y) y^{m-1} dy \\
&= \frac{1}{2^m B(1,m)} \int_{-1}^1 f\left(\frac{1}{2}(\xi+1)\right)(\xi+1)^{m-1} d\xi \\
&\approxeq \frac{1}{2^m B(1,m)} \sum_{i=0}^{n} w_i f\left(\frac{1}{2}(\xi_i+1)\right)
\end{aligned}$$

where $g_{\pi}(x)$ is the probability distribution function of $Beta(1, m)$ r.v., and the weight function in the definition of Quadrature Rule is $w(x) = (x+1)^{m-1}$. Refer to [ASR88], [Gau83] for the weights and abscissae.

Appendix C

C.1 Bitcoin price prediction using technical indicators

C.1.1 Experiment 1

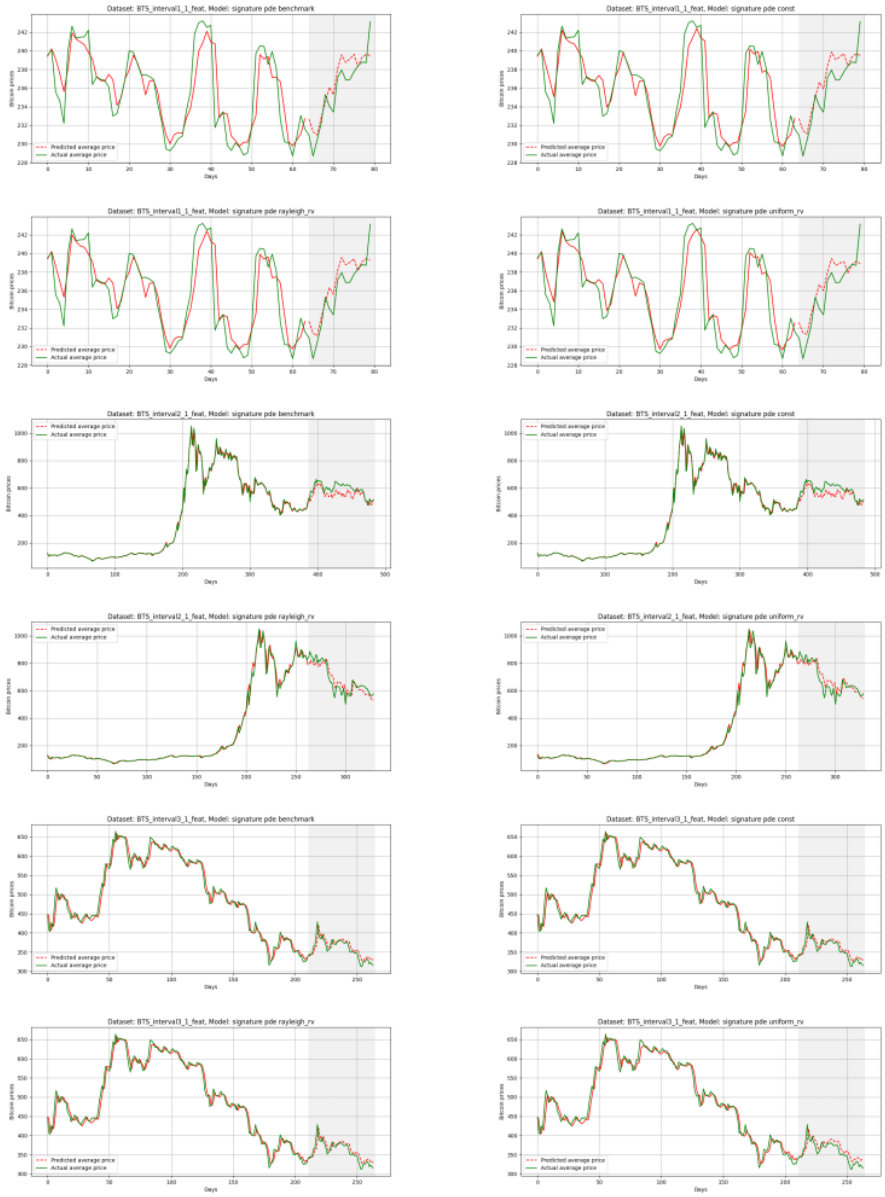


Figure C.1: Experiment 1 plots of Bitcoin price forecast for intervals I, II and III

C.1.2 Experiment 2

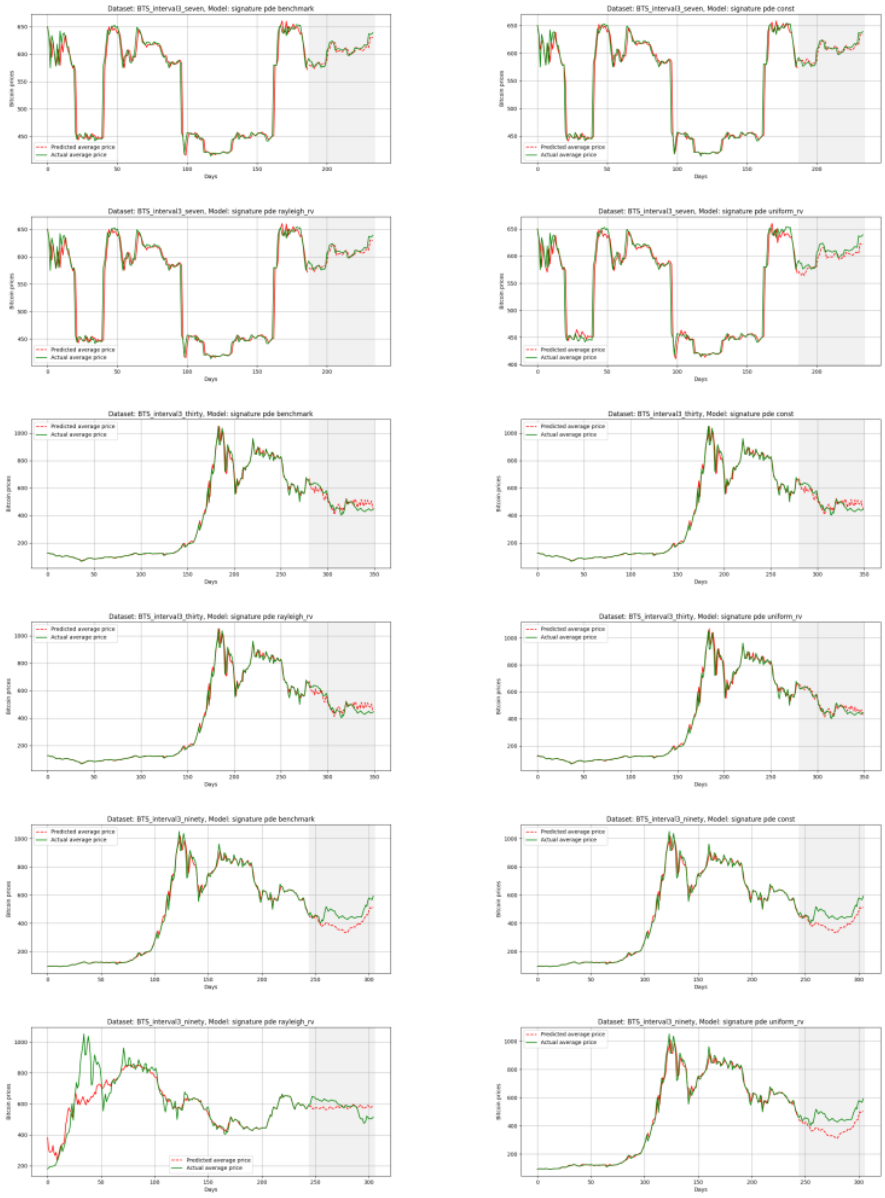
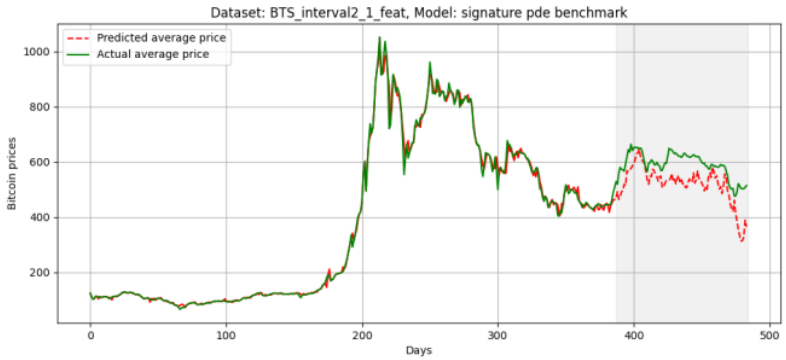


Figure C.2: Experiment 2 plots for 7, 30 and 90 days price forecast

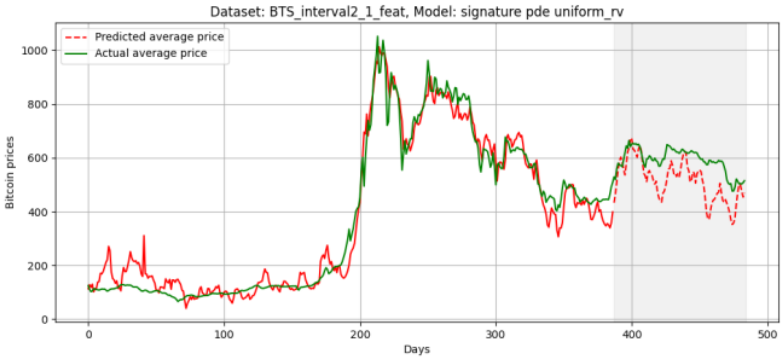
C.1.3 Experiment 3 - static kernel approximation



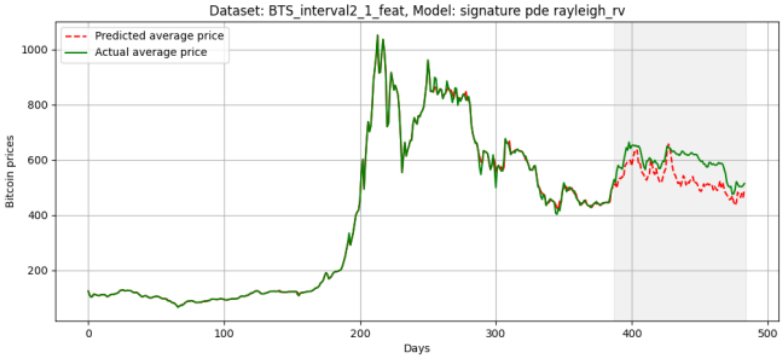
(a)

(b)

Figure C.3: Predicted vs. Actual price for *Sig-PDE* and *Const* models



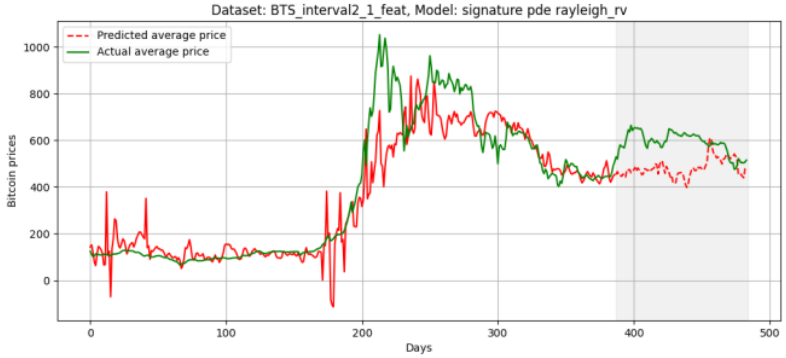
(a)



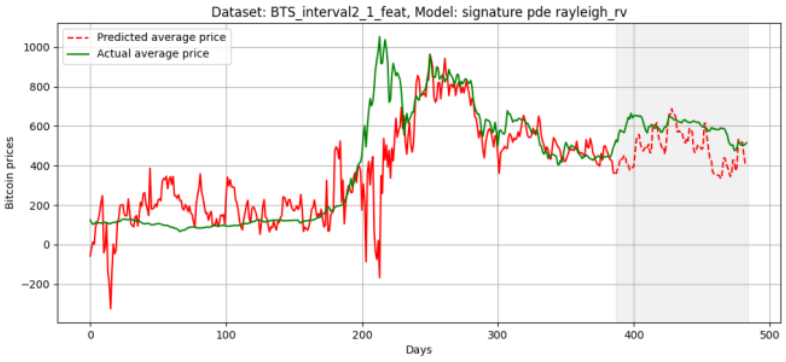
(b)

Figure C.4: Predicted vs. Actual price for *Uniform* and *Rayleigh* models

C.1.4 Experiment 4 - variations of random Fourier features for Rayleigh kernel function

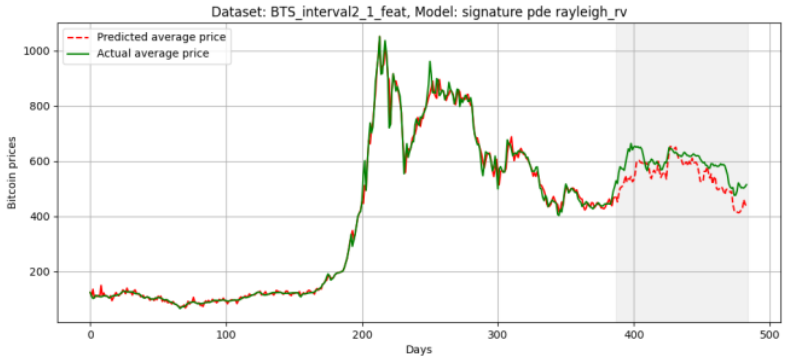


(a)

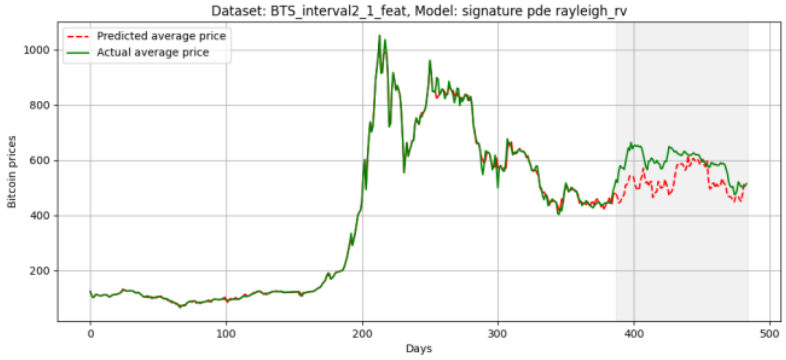


(b)

Figure C.5: Predicted vs. Actual price for *Rayleigh* kernel. (a), (b) are 3 and 6 random Fourier features respectively



(a)



(b)

Figure C.6: Predicted vs. Actual price for *Rayleigh* kernel. (a), (b) are 9 and 19 random Fourier features respectively