Embedding and learning with signatures

Adeline Fermanian

Embedding and learning with signatures

Adeline Fermanian [*]

Abstract

Sequential and temporal data arise in many fields of research, such as quantitative finance, medicine, or computer vision. The present article is concerned with a novel approach for sequential learning, called the signature method, and rooted in rough path theory. Its basic principle is to represent multidimensional paths by a graded feature set of their iterated integrals, called the signature. This approach relies critically on an embedding principle, which consists in representing discretely sampled data as paths, i.e., functions from $[0, 1]$ to $\mathbb{R}^d$. After a survey of machine learning methodologies for signatures, we investigate the influence of embeddings on prediction accuracy with an in-depth study of three recent and challenging datasets. We show that a specific embedding, called lead-lag, is systematically better, whatever the dataset or algorithm used. Moreover, we emphasize through an empirical study that computing signatures over the whole path domain does not lead to a loss of local information. We conclude that, with a good embedding, the signature combined with a simple algorithm achieves results competitive with state-of-the-art, domain-specific approaches.

1 Introduction

Sequential or temporal data are arising in many fields of research, due to an increase in storage capacity and to the rise of machine learning techniques. An illustration of this vitality is the recent relaunch of the Time Series Classification repository (Bagnall et al., 2018), with more than a hundred new datasets. Sequential data are characterized by the fact that each sample consists of an ordered array of values. Although the ordering often corresponds to time, it is not always the case. For example, text documents or DNA sequences have an intrinsic ordering, and can, therefore, be considered as sequential. Besides, when time is involved, several values can be recorded simultaneously, giving rise to an ordered array of vectors, which is, in the field of time series, often referred to as multidimensional time series. To name only a few domains, market evolution is described by financial time series, and physiological variables (e.g., electrocardiograms, electroencephalograms...) are recorded simultaneously in medicine, yielding multidimensional time series. We can also mention smartphone and GPS sensors data, or character recognition problems, where data has both a spatial and temporal aspect. These high-dimensional datasets open up new theoretical and practical challenges, as both algorithms and statistical methods need to be adapted to their sequential nature.

For the past decades, different communities have addressed this problem. First, time series forecasting has been an active area of research in statistics since the 1950s, resulting in several

[*] Sorbonne Université, CNRS, Laboratoire de Probabilités, Statistique et Modélisation, 4 place Jussieu, 75005 Paris, France, adeline.fermanian@upmc.fr

This work was supported by grants from Région Ile-de-France.

monographs, such as Hamilton (1994) or Box et al. (2015), to which we refer the reader for an overview of the domain. In essence, time series are considered as realizations of various stochastic processes, the most famous being ARIMA models. Much work in this field has been done on parameter estimation and model selection. These models have been developed for univariate time series but have been extended to the multivariate case (Lütkepohl, 2005), with the limitation that they become more complicated and harder to fit. More recently, the field of functional data analysis has extended traditional statistical methods, in particular regression and Principal Components Analysis, to functional inputs. The monographs by Ramsay and Silverman (2005) and Ferraty and Vieu (2006) will provide the reader with introductions to the area. However, methods from both time series and functional data analysis rely on strong assumptions on the regularity of the data and need to be adapted to each specific application. Therefore, modern datasets have highlighted their limitations: a lot of choices, in basis functions or model parameters, need to be handcrafted and are valid only on a small-time range. Moreover, these techniques struggle to model multidimensional series, in particular, to incorporate information about interactions between various dimensions.

On the other side, time series classification has aroused the interest of the data mining community. A broad range of algorithms have been developed, reviewed by Bagnall et al. (2017) in the univariate case. Much attention has been paid to the development of similarity measures adapted to temporal data, a popular baseline being the Dynamic Time Wrapping transform (Berndt and Clifford, 1996), followed by a 1-nearest neighbor algorithm. Bagnall et al. (2017) state that this baseline is beaten only by ensemble strategies, which combine different feature mappings. However, a great limitation of these methods is their complexity, as they have difficulty handling large time series. Recently, deep learning seems to be a promising approach and solves some problems mentioned above. For example, Fawaz et al. (2019) claim that some architectures perform systematically better than previous data mining algorithms. Nevertheless, deep learning methods are costly in memory and computing power, and often require a lot of training data.

Our goal in the present article is to discuss a novel approach for sequential learning, called the signature method, and coming from rough path theory. Its main idea is to summarize temporal or sequential inputs by the graded feature set of their iterated integrals, the signature. Note that, in rough path theory, functions are referred to as paths, to insist on their geometrical aspects. Indeed, the importance of iterated integrals had been noticed by geometers in the 60s, as presented in the seminal work of Chen (1958). It has been rediscovered by Lyons (1998) in the context of stochastic analysis and controlled differential equations, and is at the heart of rough path theory. This theory, of which Lyons et al. (2007) and Friz and Victoir (2010) give a recent account, focuses on developing a new notion of paths to make sense of evolving irregular systems. Notably, Hairer (2013) was awarded a Fields medal in 2014 for its solution to the Kardar-Parisi-Zhang equation built with rough path theory. In this context, it has been shown that the signature provides an accurate summary of a path and allows to obtain arbitrarily good linear approximations of continuous functions of paths. Therefore, assuming we want to learn an output $Y \in \mathbb{R}$, which depends on a random path $X : [0,1] \to \mathbb{R}^d$, rough path theory suggests that the signature is a relevant feature set to describe X.

As can be expected, the signature has recently received the attention of the machine learning community and has achieved a series of successful applications. To cite some of them, Yang et al. (2016) have achieved state-of-the-art results for handwriting recognition with a recurrent neural network combined with signature features. Graham (2013) have used the same approach for character recognition, and Gyurkó et al. (2014) have coupled Lasso with signature features for financial data streams classification. Kormilitzin et al. (2016) have investigated its use for the detection of bipolar disorders, and Yang et al. (2017) for human action recognition.

For a gentle introduction to the signature method in machine learning, we refer the reader to Chevyrev and Kormilitzin (2016).

However, despite many promising empirical successes, a lot of questions remain open, both practical and theoretical. In particular, to compute the signature, it is necessary to embed discretely sampled data points into paths. While authors use different approaches, this embedding is only mentioned in some articles, and rarely discussed. Thus, our purpose in this paper is to take a step forward in understanding how signature features should be constructed for machine learning tasks, with a special focus on the embedding step. Our document is organized as follows.

(i) First, in Section 2, we give a brief exposition of the signature definition and properties, along with a survey of different approaches undertaken in the literature to combine signatures with machine learning algorithms. Datasets used throughout the paper are also presented in Section 3.

(ii) Then, we review in Section 4 potential embeddings and compare their predictive performance. We emphasize that the embedding is as a crucial step as the algorithm choice since it can drastically change accuracy results. Moreover, we point out that one embedding, called lead-lag, performs systematically better than others, and this consistently other different datasets and learning algorithms.

(iii) Furthermore, we investigate in Section 5 the choice of signature domain. Indeed, signatures can be computed on any sub-interval of the path definition domain, and it is natural to wonder whether some local information is lost when signatures of the whole path are computed. We end the section by showing that, with a good embedding, the signature combined with a simple algorithm, such as a random forest classifier, obtains results comparable to state-of-the-art approaches in different application areas, while remaining a generic approach and computationally simple.

(iv) Finally, we discuss in Section 6 some open questions that we find worth investigating in future work.

Our empirical results are based on three recent datasets, in different fields of application. One is a univariate sound recording dataset, called Urban Sound (Salamon et al., 2014), whereas the others are multivariate. One has been made available by Google (2017), and consists of drawing trajectories, while the other is made up of 12 channels recorded from smartphone sensors (Malekzadeh et al., 2018). They are each of a different nature and present a variety of lengths, noise levels, and dimensions. In this way, we intend to get generic results, independent of particular domain-specific aspects of the data. The code generating results of Sections 4 and 5 is available at `https://github.com/afermanian/embedding_with_signatures`.

2 A first glimpse of the signature method

2.1 Definition and main properties

We introduce in this subsection the notion of signature and review some of its important properties. The reader is referred to Lyons et al. (2007) or Friz and Victoir (2010) for a more involved mathematical treatment with proofs. Throughout the article, our basic objects are paths, that is functions from $[0, 1]$ to $\mathbb{R}^d$, where $d \in \mathbb{N}^*$. The main assumption is that these paths are of bounded variation, i.e., they have finite length.

Definition 1 *Let*

$$\begin{aligned} X : [0,1] &\longrightarrow \mathbb{R}^d \\ t &\longmapsto (X_t^1, \dots, X_t^d). \end{aligned}$$

The total variation of X is defined by

$$\|X\|_{1-var} = \sup_D \sum_{t_i \in D} \|X_{t_i} - X_{t_{i-1}}\|,$$

where the supremum is taken over all finite partitions

$$D = \big\{(t_0, \dots, t_k) \,|\, k \geq 1, 0 = t_0 < t_1 < \cdots < t_{k-1} < t_k = 1\big\}$$

of $[0,1]$, and $\|\cdot\|$ denotes the Euclidean norm on $\mathbb{R}^d$. The path X is said to be of bounded variation if its total variation is finite.

The set of bounded variation paths is exactly the set of functions whose first derivatives exist almost everywhere. Being of bounded variation is therefore not a particularly restrictive assumption. It contains, for example, all Lipschitz functions. In particular, if X is continuously differentiable, and $\dot{X}$ denotes its first derivative with respect to t, then

$$\|X\|_{1-var} = \int_0^1 \|\dot{X}_t\| dt.$$

The assumption of bounded variation allows to define Riemann-Stieljes integrals along paths. We do not give a detailed exposition of this integration theory here, but we refer the interested reader to Lyons et al. (2007). From now on, we assume that the integral of a continuous path $Y : [0,1] \to \mathbb{R}^d$ against a path of bounded variation $X : [0,1] \to \mathbb{R}^d$ is well-defined on any $[s,t] \subset [0,1]$, and denoted by

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

where $X = (X^1, \dots, X^d)$, and $Y = (Y^1, \dots, Y^d)$. When X is continuously differentiable, this integral is equal to the standard Riemann integral, that is,

$$\int_s^t Y_u dX_u = \int_s^t Y_u \dot{X}_u du.$$

As an example, assume that X is linear, i.e.,

$$X_t = (X_t^1, \dots, X_t^d) = (a_1 + b_1 t, \dots, a_d + b_d t), \quad 0 \leq t \leq 1, \quad a_1, \dots, a_d, b_1, \dots b_d \in \mathbb{R}. \tag{1}$$

Then,

$$\int_s^t dX_u = \int_s^t \dot{X}_u du = \begin{pmatrix} \int_s^t b_1 du \\ \vdots \\ \int_s^t b_d du \end{pmatrix} = \begin{pmatrix} b_1(t-s) \\ \vdots \\ b_d(t-s) \end{pmatrix}.$$

The formula above is useful, since in practice we only compute the integral of linear paths, as discussed later in this subsection. We are now in a position to define the signature.

Definition 2 *Let $X : [0,1] \to R^d$ be a path of bounded variation, $I = (i_1, \dots, i_k) \subset \{1, \dots d\}^k$, $k \in \mathbb{N}^*$, be a multi-index of length k, and $[s,t] \subset [0,1]$ be an interval. The signature coefficient of X corresponding to the index I on $[s,t]$ is defined by*

$$S^I(X)_{[s,t]} = \underset{s \le u_1 < \cdots < u_k \le t}{\int \cdots \int} dX^{i_1}_{u_1} \dots dX^{i_k}_{u_k} = \int_s^t \Big(\int_{u_1}^t \Big(\int_{u_2}^t \cdots \int_{u_{k-1}}^t dX^{i_k}_{u_k} \Big) dX^{i_2}_{u_2} \Big) dX^{i_1}_{u_1}. \tag{2}$$

$S^I(X)_{[s,t]}$ is then said to be a signature coefficient of order k.

The signature of X is the sequence containing all signature coefficients, i.e.,

$$S(X)_{[s,t]} = \big(1, S^{(1)}(X)_{[s,t]}, \dots, S^{(d)}(X)_{[s,t]}, S^{(1,1)}(X)_{[s,t]}, S^{(1,2)}(X)_{[s,t]}, \dots, S^{(i_1, \dots, i_k)}(X)_{[s,t]}, \dots \big).$$

The signature of X truncated at order K, denoted by $S^K(X)$, is the sequence containing all signature coefficients of order lower than or equal to K, that is

$$S^K(X)_{[s,t]} = \big(1, S^{(1)}(X)_{[s,t]}, S^{(2)}(X)_{[s,t]}, \dots, S^{\overbrace{(d, \dots, d)}^{K}}(X)_{[s,t]}\big).$$

For simplicity, when $[s,t] = [0,1]$, we omit the interval in the notations, and, e.g., write $S^K(X)$ instead of $S^K(X)_{[0,1]}$. Before giving an example of signature, some comments are in order. First, we note that, for a path in $\mathbb{R}^d$, there are d^k coefficients of order k. The signature truncated at order K is therefore a vector of dimension

$$\sum_{k=0}^{K} d^k = \frac{d^{K+1} - 1}{d - 1} \quad \text{if } d \neq 1,$$

and $K + 1$ if $d = 1$. Unless otherwise stated, we assume that $d \neq 1$, as this is in practice usually the case. Thus, the size of $S^K(X)$ increases exponentially with K, and polynomially with d—some typical values are presented in Table 1. The set of coefficients of order k can be seen as an element of the kth tensor product of $\mathbb{R}^d$ with itself, denoted by $(\mathbb{R}^d)^{\otimes k}$. For example, we can write the d coefficients of order 1 as a vector, and the d^2 coefficients of order 2 as a matrix, i.e.,

$$\begin{pmatrix} S^{(1)}(X) \\ \vdots \\ S^{(d)}(X) \end{pmatrix} \in \mathbb{R}^d, \quad \begin{pmatrix} S^{(1,1)}(X) & \dots & S^{(1,d)}(X) \\ \vdots & & \vdots \\ S^{(d,1)}(X) & \dots & S^{(d,d)}(X) \end{pmatrix} \in \mathbb{R}^{d \times d} \approx (\mathbb{R}^d)^{\otimes 2}.$$

Similarly, coefficients of order 3 can be written as a tensor of order 3, and so on. Then, $S(X)$ can be seen as an element of the tensor algebra

$$\mathbb{R} \oplus \mathbb{R}^d \oplus (\mathbb{R}^d)^{\otimes 2} \oplus \cdots \oplus (\mathbb{R}^d)^{\otimes k} \oplus \cdots .$$

This structure of tensor algebra will not be used in the present article but is extremely useful to derive properties of the signature (Lyons, 1998; Friz and Victoir, 2010; Hambly and Lyons, 2010).

Finally, it should be noted that, due to the ordering in the integration domain in (2), signature coefficients are not symmetric. For example, $S^{(1,2)}(X)$ is a priori not equal to $S^{(2,1)}(X)$.

As a toy example, let us consider the linear path (1) again, and assume for simplicity that $d = 2$:

$$X_t = \begin{pmatrix} X^1_t \\ X^2_t \end{pmatrix} = \begin{pmatrix} a_1 + b_1 t \\ a_2 + b_2 t \end{pmatrix}.$$

	$d=2$	$d=3$	$d=6$
$K=1$	2	3	6
$K=2$	6	12	42
$K=5$	62	363	9330
$K=7$	254	3279	335922

Table 1: Typical sizes of $S^K(X)$ for different values of K and d, where $X : [0,1] \to \mathbb{R}^d$.

Then, for any $[s,t] \subset [0,1]$ the signature coefficients of order 1 are

$$S^{(1)}(X)_{[s,t]} = \int_s^t dX_u^1 = b_1(t-s) \quad \text{and} \quad S^{(2)}(X)_{[s,t]} = \int_s^t dX_u^2 = b_2(t-s).$$

The first coefficient of order 2 is

$$S^{(1,1)}(X)_{[s,t]} = \int_s^t \int_{u_1}^t dX_{u_2}^1 dX_{u_1}^1 = \int_s^t \int_{u_1}^t b_1^2 du_2 du_1 = b_1^2 \int_s^t (t-u_1) du_1 = \frac{b_1^2(t-s)^2}{2}.$$

Similarly,

$$S^{(1,2)}(X)_{[s,t]} = S^{(2,1)}(X) = \frac{b_1 b_2 (t-s)^2}{2} \quad \text{and} \quad S^{(2,2)}(X)_{[s,t]} = \frac{b_2^2(t-s)^2}{2}.$$

For any index $I = (i_1, \dots, i_k) \subset \{1,2\}^k$, we easily obtain

$$S^{(i_1,\dots,i_k)}(X)_{[s,t]} = \underset{s \le u_1 < \dots < u_k \le t}{\int \cdots \int} dX_{u_1}^{i_1} \dots dX_{u_k}^{i_k} = \frac{b_{i_1} \dots b_{i_k} (t-s)^k}{k!}. \tag{3}$$

A crucial feature of the signature is that it encodes geometric properties of the path. Indeed, it is clear that coefficients of order 2 correspond to some areas outlined by the path, as shown in Figure 1. For higher orders of truncation, the signature contains information about the joint evolution of tuples of coordinates (see, e.g., Yang et al., 2017). Furthermore, the signature possesses several properties that make it a good statistical summary of paths, as shown in the next four propositions.

Proposition 1 *Let $X : [0,1] \to \mathbb{R}^d$ be a path of bounded variation, and $\psi : [0,1] \to [0,1]$ be a non-decreasing surjection. Then, if $\widetilde{X}_t = X_{\psi(t)}$ is the reparametrization of X under ψ,*

$$S(\widetilde{X}) = S(X).$$

This proposition is a consequence of the properties of integrals and bounded variation paths (Friz and Victoir, 2010, Proposition 7.10). In other words, the signature of a path is the same up to any reasonable time change. There is, therefore, no information about the path travel time in signature coefficients. However, when relevant for the application, it is possible to include this information by adding the time parametrization as a coordinate of the path. This procedure plays a crucial role in the construction of time embeddings, which will be thoroughly discussed in Section 4.

A second important property is a condition ensuring uniqueness of signatures.

Proposition 2 *If X has at least one monotonous coordinate, then $S(X)$ determines X uniquely.*

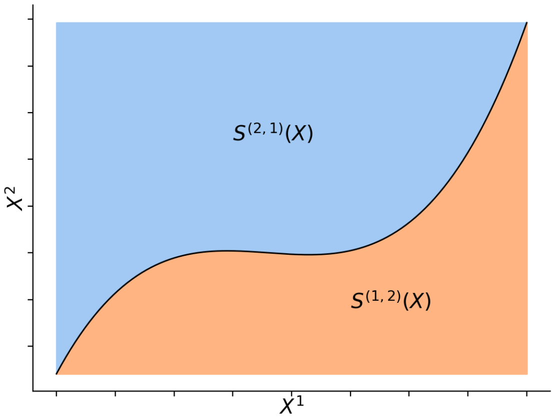


Figure 1: Geometric interpretation of signature coefficients.

It should be noticed that having a monotonous coordinate is a sufficient condition, but a necessary one can be found in the monograph by Hambly and Lyons (2010), together with a proof of the proposition. The principal significance of this result is that it provides a practical procedure to guarantee signature uniqueness: it is sufficient to add a monotonous coordinate to the path X. For example, the time embedding mentioned above will satisfy this condition.

This result does not provide a practical procedure to reconstruct a path from its signature. However, this is an active area of research (Chang et al., 2017; Lyons and Xu, 2017, 2018). In particular, Lyons and Xu (2017) derive an explicit expression of rectilinear paths, defined in Section 4.1, in terms of their signatures; and Lyons and Xu (2018) construct, from the signature of a $\mathcal{C}^1$ path, a sequence of piecewise linear approximations converging to the initial path.

The next proposition reveals that the signature linearizes functions of X. We refer the reader to Király and Oberhauser (2019, Theorem 1) for a proof.

Proposition 3 *Let D be a compact subset of the space of bounded variation paths from $[0,1]$ to $\mathbb{R}^d$. Let $f : D \to \mathbb{R}$ be continuous. Then, for every $\epsilon > 0$, there exists $N \in \mathbb{N}$, $w \in \mathbb{R}^N$, such that, for any $X \in D$,*

$$\left| f(X) - \langle w, S(X) \rangle \right| \leq \epsilon,$$

where $\langle \cdot, \cdot \rangle$ denotes the Euclidean scalar product on $\mathbb{R}^N$.

This proposition is a consequence of the Stone-Weierstrass theorem. The classical Weierstrass approximation theorem states that every real-valued continuous function on a closed interval can be uniformly approximated by a polynomial function. Similarly, we obtain here that any real-valued continuous function on a compact subset D of bounded variation paths can be uniformly approximated by a linear form on the signature. Linear forms on the signature can, therefore, be thought of as the equivalent of polynomial functions for paths.

Finally, Chen's theorem (Chen, 1958) provides a formula to compute recursively the signature of a concatenation of paths. Let $X : [s,t] \to \mathbb{R}^d$ and $Y : [t,u] \to \mathbb{R}^d$ be two paths, $0 \leq s < t < u \leq 1$, the concatenation of X and Y, denoted by $X * Y$, is defined as the path from $[s,u]$ to $\mathbb{R}^d$ such that, for any $v \in [s,u]$,

$$(X * Y)_v = \begin{cases} X_v & \text{if } v \in [s,t], \\ X_t + Y_v - Y_t & \text{if } v \in [t,u]. \end{cases}$$

Proposition 4 (Chen) *Let $X : [s,t] \to \mathbb{R}^d$ and $Y : [t,u] \to \mathbb{R}^d$ be two paths with bounded variation. Then, for any multi-index $(i_1, \dots, i_k) \subset \{1, \dots, d\}^k$,*

$$S^{(i_1,\dots,i_k)}(X * Y) = \sum_{\ell=0}^{k} S^{(i_1,\dots,i_\ell)}(X) \cdot S^{(i_{\ell+1},\dots,i_k)}(Y). \tag{4}$$

This proposition is an immediate consequence of the linearity property of integrals (Lyons et al., 2007, Theorem 2.9). However, it is essential for the explicit calculation of signatures. Indeed, in practice, since we are given a finite number of observations sampled from X, we must interpolate these points, which yields a continuous piecewise linear path. To compute its signature, it is then sufficient to iterate the following two steps:

1. Compute with equation (3) the signature of a linear section of the path.

2. Concatenate it to the other pieces with Chen's formula (4).

This procedure is implemented in the Python library `iisignature` (Reizenstein and Graham, 2018). Thus, for a sample consisting of p points in $\mathbb{R}^d$, if we consider the path formed by their linear interpolation, the computation of the path signature truncated at level K takes $O(pd^K)$ operations. The complexity is therefore linear in the number of sampled points but exponential in the truncation order K. Notice that the size of the signature vector is also exponential in the truncation order K, as shown in Table 1. Therefore, in applications, K is constrained to remain small, typically of order less than 10.

2.2 Signature and machine learning

Now that we have presented the signature and its properties, we focus on its use in machine learning. In this context, we place ourselves in a statistical framework, and assume that our goal is to understand the relationship between a random input path $X : [0,1] \to \mathbb{R}^d$ and a random output $Y \in \mathbb{R}$. In a classical setting, we would be given a sample of independent and identically distributed (i.i.d.) observations $\{(X_1, Y_1), \dots, (X_n, Y_n)\}$, drawn from (X, Y). However, in applications, we only observe a realization X_i sampled at a discrete set of times $0 \leq t_1 < \cdots < t_{p_i} \leq 1$, $p_i \in \mathbb{N}^*$. Therefore, we are given an i.i.d. sample $\{(\mathbf{x_1}, Y_1), \dots, (\mathbf{x_n}, Y_n)\}$, where $\mathbf{x_i}$ takes the form of a matrix, i.e.,

$$\mathbf{x_i} = \begin{pmatrix} x_{i,1}^1 & \cdots & x_{i,p_i}^1 \\ \vdots & & \vdots \\ x_{i,1}^d & \cdots & x_{i,p_i}^d \end{pmatrix} \in \mathbb{R}^{d \times p_i}. \tag{5}$$

In this notation, $x_{i,j}^k$ denotes the kth coordinate of the ith sample observed at time t_j. If $d = 1$, we are in a classical setting of time series, where each sample is sampled in a finite number of points. However, d may here differ from 1, so we find ourselves in a more general situation where we want to learn from multidimensional time series. Moreover, it is worth noting the dependence of the number of sampled points p_i on i. In other words, each observation may have a different length. The signature dimension being independent of the number of sampled points, representing time series by their signature naturally handles inputs of various lengths, whereas traditional methods often require them to be normalized to a fixed length. To sum up, the signature method is appropriate for learning with discretely sampled multidimensional time series, possibly of different lengths.

As an example, we consider the Google dataset Quick, Draw! (Google, 2017). It consists of pen trajectories of millions of drawings, divided into 340 classes. Some samples are drawn in Figure 2. In this case, we see that the y_i's are discrete labels of the drawing's class, and the $\mathbf{x_i}$'s are matrices of pen coordinates. Therefore, in this example, $d = 2$ and p_i varies for each drawing, but is typically in the order of a few dozen points.

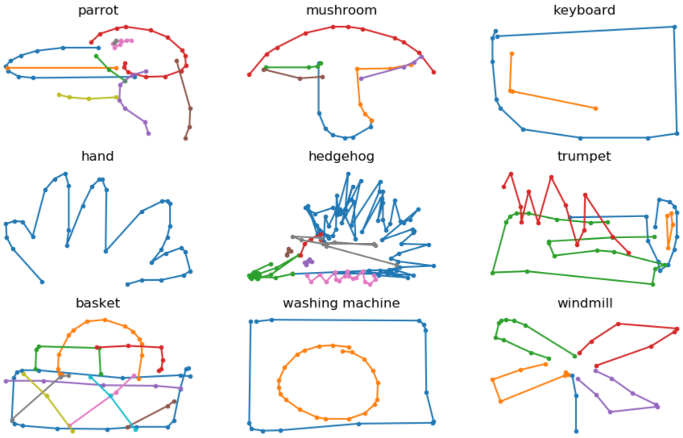


Figure 2: 9 drawings from the Quick, Draw! dataset

As discussed in the introduction, to use signature features, one needs to embed the observations $\mathbf{x_i}$ into paths of bounded variation $X_i : [0,1] \to \mathbb{R}^d$. This step, which also consists of adding other coordinates, such as time, will be thoroughly discussed in Section 4. Therefore, we assume for the moment that we are given a set of embeddings X_i, $1 \leq i \leq n$, from which signature features can be computed. When an embedding has been chosen, one can compute signature features and use them in combination with a learning algorithm. The procedure can be summarized as follows:

$$\text{Raw data} \longrightarrow \text{Embedding} \longrightarrow \text{Signature features} \longrightarrow \text{Algorithm.}$$

The literature on the combination of signature features with learning algorithms can be divided into three groups. These groups correspond to the nature of the algorithm's input: it is either a vector, a sequence or an image. In the context of deep learning, this division matches the different classes of neural network architectures: feedforward, recurrent and convolutional networks. The latter only deals with input paths in $\mathbb{R}^2$, with applications such as characters or handwriting recognition.

The first approach is to compute the signature of X on its whole domain, that is on $[0,1]$. In this way, X is mapped into a finite set of coefficients, that is then fed into a predictive algorithm, typically a feedforward neural network. This strategy is implemented by Yang et al. (2017) for skeleton-based human action recognition. From a sequence of human joints' positions, the authors construct a high dimensional vector of signature coefficients, which is then the input of a small dense network. Gyurkó et al. (2014); Lyons et al. (2014) also apply this method to financial time series, combining it with Lasso and OLS regression. The procedure is illustrated in Figure 3.

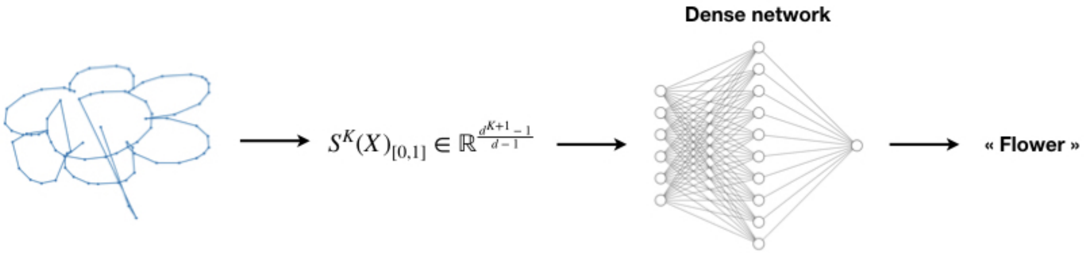


Figure 3: Signature and dense neural network.

A second family of methods consists in describing the input path by a sequence of signature coefficients. There are several variants of this approach, and we present here in detail the one chosen by Wilson-Nunn et al. (2018) for Arabic handwriting recognition, for its simplicity and representativeness. To create a signature sequence, the time interval $[0, 1]$ is divided into a dyadic partition

$$0 \leq 2^{-q} < \cdots < j2^{-q} < \cdots < (2^q - 1)2^{-q} \leq 1, \tag{6}$$

where $q \in \mathbb{N}$. By computing the signature truncated at order K on every dyadic interval $[j2^{-q}, (j+1)2^{-q}]$, $0 \leq j < 2^q$, we obtain a sequence of 2^q signature vectors, each of dimension $(d^{K+1} - 1)/(d-1)$. This sequence is typically fed into a recurrent network, as illustrated in Figure 4. We recall that recurrent networks are a class of neural networks that handle sequential inputs by recursively applying the same transformations to elements of a sequence but keeping in memory previous computations, so that dependence between consecutive elements is taken into account. In our case, the whole approach boils down to transforming the original sequential data into another sequence of signature coefficients. Such a procedure may be surprising, as the original data could have been itself the input of a recurrent network, instead of being mapped into a new signature sequence. Moreover, in the process, the dimension of the input of the recurrent network has been increased from d to $(d^{K+1} - 1)/(d-1)$. However, Wilson-Nunn et al. (2018) show the superiority of this approach and several other authors have successfully carried out this type of method. For example, Lai et al. (2017) achieve state-of-the-art results in the problem of recognizing genuine signatures from forgeries, and Liu et al. (2017) of distinguishing different writers. In both cases, they use a sliding window over the character rather than a dyadic partition of the time interval, but otherwise, have an approach similar to Wilson-Nunn et al. (2018).

Finally, a third group of authors have taken these ideas further and created images of signature coefficients. Their rationale is to mix temporal and pictorial aspects of the data. Indeed, assuming that the input path is a trajectory in $\mathbb{R}^2$, it can be turned into an image by forgetting its temporal aspect and setting the pixel values to 1 along the trajectory, and 0 elsewhere. Then, starting from this representation, a bunch of images are created, such that each image corresponds to a signature coefficient, as shown in Figure 5. This can be done in various ways. However, the general idea is to consider a sliding window following the path and to set the pixel value at the center of the window to be equal to the signature coefficient

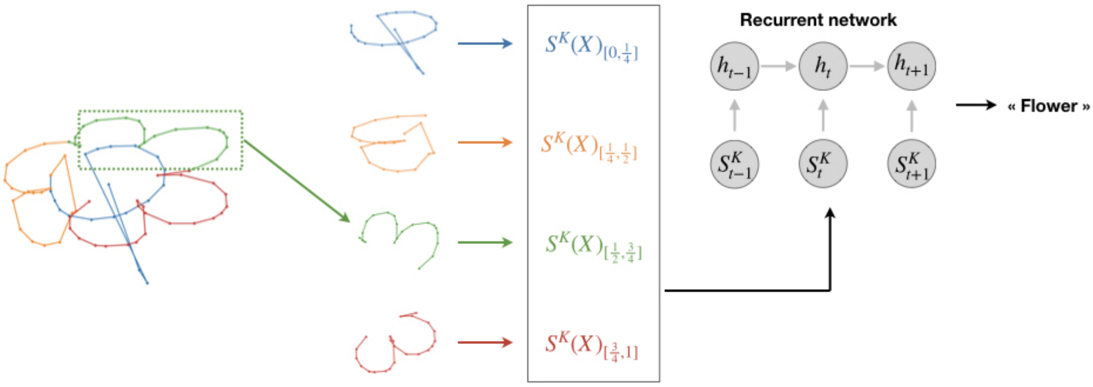

$S^K(X)_{[0,\frac{1}{4}]}$
$S^K(X)_{[\frac{1}{4},\frac{1}{2}]}$
$S^K(X)_{[\frac{1}{2},\frac{3}{4}]}$
$S^K(X)_{[\frac{3}{4},1]}$
Recurrent network
h_{t-1}
h_t
h_{t+1}
S^K_{t-1}
S^K_t
S^K_{t+1}
« Flower »

Figure 4: Signature and recurrent neural network.

computed over the window. This procedure is repeated by shifting the window by 1, as shown in Figure 6 for times t_4, t_5 and t_6. If the signature is truncated at order K, this yields $2^{K+1} - 1$ sparse gray pictures, which can then be the input of a convolutional neural network. Graham (2013) and Yang et al. (2016, 2015) have obtained significant accuracy improvements for character recognition and writer identification with this approach.

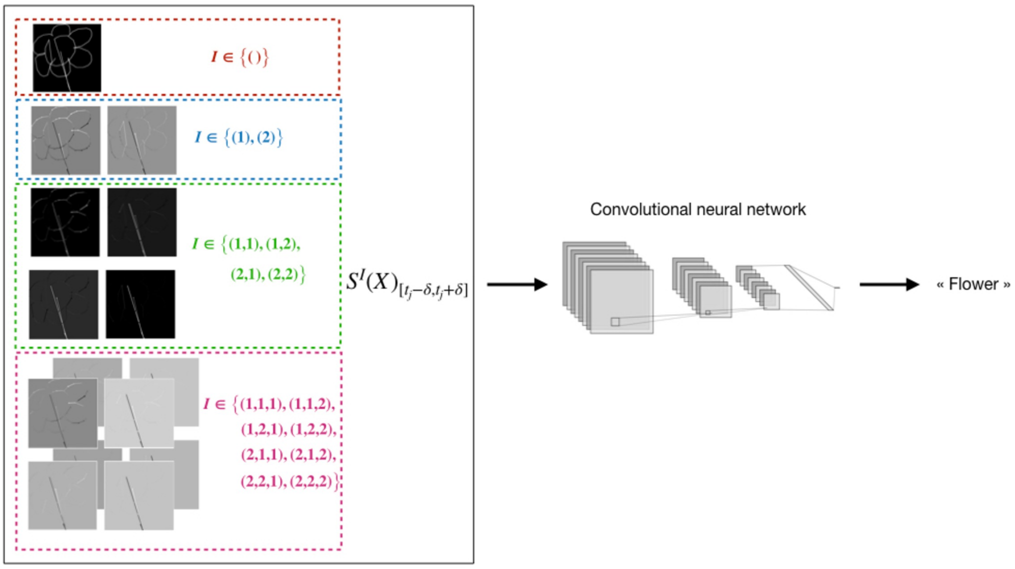


Figure 5: Signature and convolutional neural network.

To sum up, we see that the signature may be used in various ways, and for different applications. Several points of view coexist, and none of them has shown to be systematically better. In particular, on the one hand, signatures may be used to remove temporal aspects and

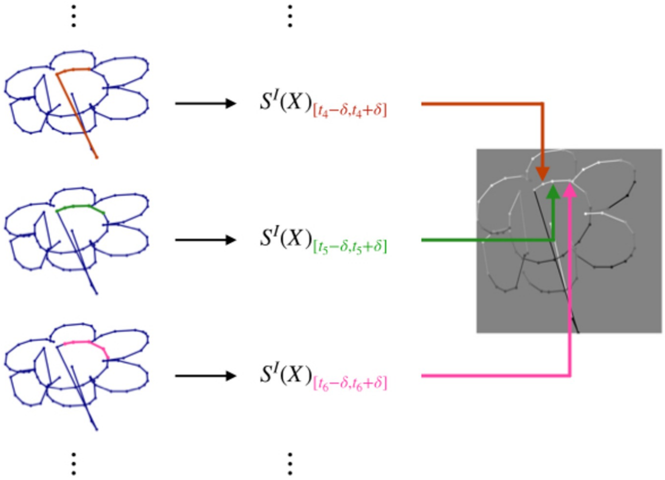

$S^I(X)_{[t_4-\delta,t_4+\delta]}$
$S^I(X)_{[t_5-\delta,t_5+\delta]}$
$S^I(X)_{[t_6-\delta,t_6+\delta]}$

Figure 6: Construction of an image with signature coefficients.

to reduce the dimension of the problem, whereas, on the other hand, they may do the opposite and increase the dimension of the algorithm's input. Moreover, they are combined with various learning algorithms and it may be hard to distinguish the properties of the signature from those of the algorithms. Nevertheless, all these methods assume that discrete data points have been embedded into actual continuous paths. As we will see in Section 4, the choice of path is crucial. Therefore, we describe in the next section the datasets used throughout the article, to understand their underlying structure and find suitable embeddings.

3 Datasets

The datasets used in this article have been chosen to cover a broad range of applications while being recent and challenging in various ways. Moreover, they present a variety of sampling frequencies and dimensions. They illustrate therefore different potential embeddings.

First, the Quick, Draw! dataset (Google, 2017), which was already discussed in Section 2.2, and illustrated in Figure 2, is a public Google dataset. It consists of 50 million drawings, each drawing being a sequence of time-stamped pen stroke trajectories, divided into 340 categories. It takes approximately 7 gigabytes of hard disk space and is, therefore, a particularly large dataset. To compute the signature of every sample, it would thus be necessary to design a specific architecture, which cannot be implemented on a standard laptop computer. However, our goal in the present article is not to achieve the best possible performance, but to understand embedding properties. Moreover, we would like our experiments to be easily reproducible without requiring a lot of computational capacities. Therefore, we choose to use only a subset of the data. In Sections 4.2 and 5.1 we will use 68 000 training examples, while in Section 5.2, to improve accuracy, we will take around 12 million training samples.

Let us describe more precisely the data format. When someone draws an object, two pieces of information are recorded: pen positions, sampled at different times, and pen jumps. Therefore, one drawing consists of a set of strokes, one stroke being a segment of the drawing between two pen jumps, represented with different colors in Figure 2. As each stroke can be of different length, if $p_{i,k}$ is the number of points in the kth stroke of drawing number i, and if this drawing has K_i strokes, then one drawing consists of K_i tables of sizes $2 \times p_{i,1}, \dots, 2 \times p_{i,K_i}$, where the factor 2 corresponds to the plane $\mathbb{R}^2$. For example, in Figure 2, the windmill drawing has 5 strokes: the first stroke is the blue one with 3 points, the second one the orange with 5 points, and so on. The data has been preprocessed by Google, resulting in the so-called "simplified drawing files". The preprocessing details are not given here, but we refer the reader to Google (2017) for a complete description. Finally, each sample i can be encoded under the following compact form:

$$\mathbf{x_i} = \begin{pmatrix} x^1_{i,1} & \dots & x^1_{i,p_{i,1}} & \dots & x^1_{i,p_{i,1}+\cdots+p_{i,K-1}+1} & \dots & x^1_{i,p_{i,1}+\cdots+p_{i,K}} \\ x^2_{i,1} & \dots & x^2_{i,p_{i,1}} & \dots & x^2_{i,p_{i,1}+\cdots+p_{i,K-1}+1} & \dots & x^2_{i,p_{i,1}+\cdots+p_{i,K}} \\ 1 & \dots & 1 & \dots & K_i & \dots & K_i \end{pmatrix} \in \mathbb{R}^{3 \times p_i}, \quad (7)$$

where $(x^1_{i,j}, x^2_{i,j})$ are the coordinates of the jth point of the drawing number i, and there is a pen jump when the last row of $\mathbf{x_i}$ increases by 1. As in (5), $p_i = p_{i,1} + \cdots + p_{i,K}$ denotes the total number of points of drawing i.

The second dataset we are interested in is the Urban Sound dataset (Salamon et al., 2014). It consists of sound recordings, divided into 10 classes: car horn, dog barking, air conditioner, children playing, drilling, engine idling, gunshot, jackhammer, siren, and street music. It contains both mono and stereo recordings, so some samples take values in $\mathbb{R}$, and some in $\mathbb{R}^2$. By averaging the two channels of stereo sounds, the data has been normalized to mono recordings, so that each sample is a one-dimensional time series. We thus have a collection of 5 435 time series of various lengths. On average, they are sampled at approximately 170 000 points, which makes them long time series, typically hard to model along the whole time range. Figure 7 depicts some examples of these noisy time series.

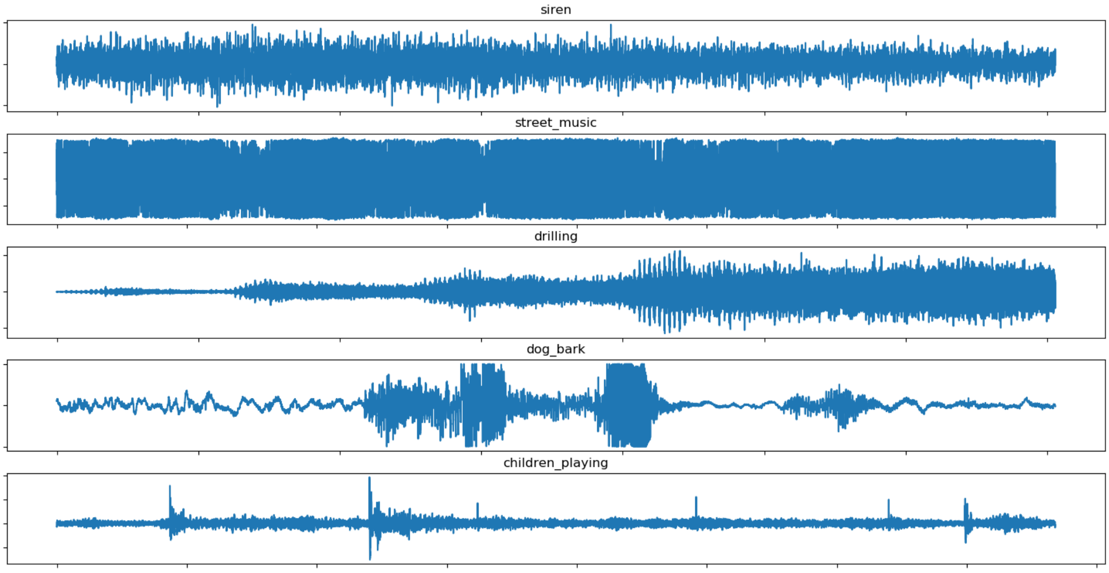


Figure 7: 5 samples from the Urban sound dataset

Finally, we consider the MotionSense dataset, composed of smartphone sensory data generated by accelerometer and gyroscope sensors (Malekzadeh et al., 2018). This data has been recorded while some participants performed an activity among walking upstairs and downstairs, walking, jogging, sitting, and standing. In total, there are 10 classes and 360 recordings, which correspond to 24 participants performing 15 different trials. During each trial, 12 variables are measured: 3 directions for attitude, gravity, user acceleration, and rotation rate, respectively. Information about the participants is provided but we focus on the task of recognizing the activity performed from the multidimensional time series formed by sensors data. In Figure 8, three samples are shown, and, for each of them, curves of different colors correspond to the various quantities measured by sensors. Therefore, every sample is a time series in $\mathbb{R}^{12}$ of different lengths. It is clear from Figure 8 that these series are noisy and highly dimensional. We can also note that they are shorter than the Urban sound's ones, with an average of approximately 4 000 time steps.

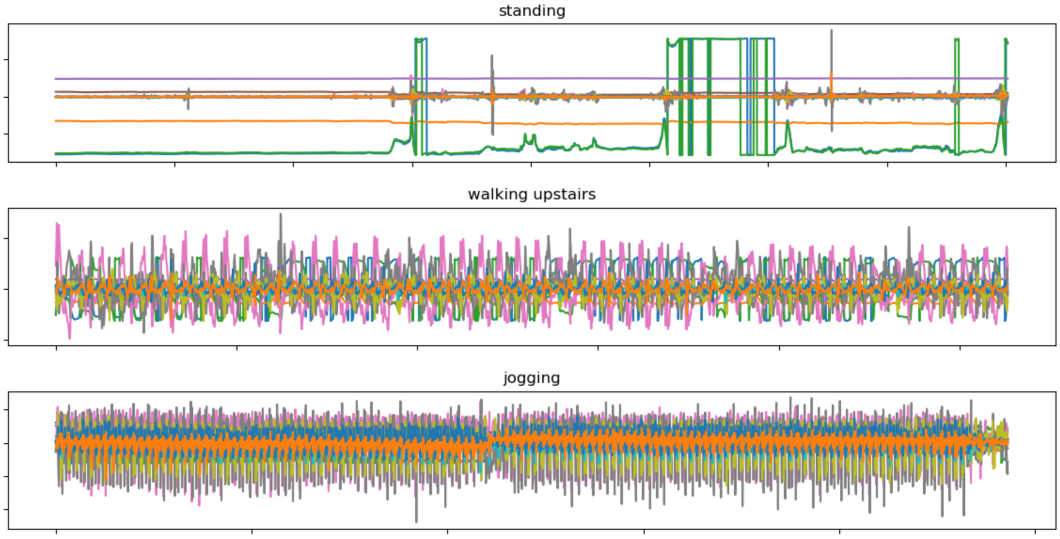


Figure 8: 3 samples from the Motion sense dataset

We summarize in Table 2 some characteristics of these three datasets. They illustrate the diversity of problems in sequential learning, where time appears in different ways.

	Quick, Draw!	Urban Sound	Motion Sense
Number of classes	340	10	6
Dimension	2	1	12
Average number of sampled points	44	171 135	3 924
Training set size	68 000	4 435	300
Validation set size	6 800	500	30
Test set size	6 800	500	30

Table 2: Datasets summary

4 The embedding

We are now in a position to conduct a study of embeddings. In practice, we have at our disposal a matrix of observations $\mathbf{x_i} \in \mathbb{R}^{d \times p_i}$, written in (5), where columns correspond to points in $\mathbb{R}^d$ sampled at times $0 \leq t_1 < \cdots < t_{p_i} \leq 1$. As already explained in Section 2.2, the goal is to construct a continuous path $X_i : [0, 1] \to \mathbb{R}^d$ from the matrix $\mathbf{x_i}$. Therefore, we need to choose an interpolation method, but, to ensure some properties such as signature uniqueness (see Proposition 2), we may also create new coordinates to the path and in this way increase the dimension d of the embedding space. When not hidden, the embedding is generally only mentioned in the literature, without in-depth discussions. Therefore, our purpose here is not only to compare embeddings' performance, but also to give a first systematic survey of their use in the context of learning with signatures.

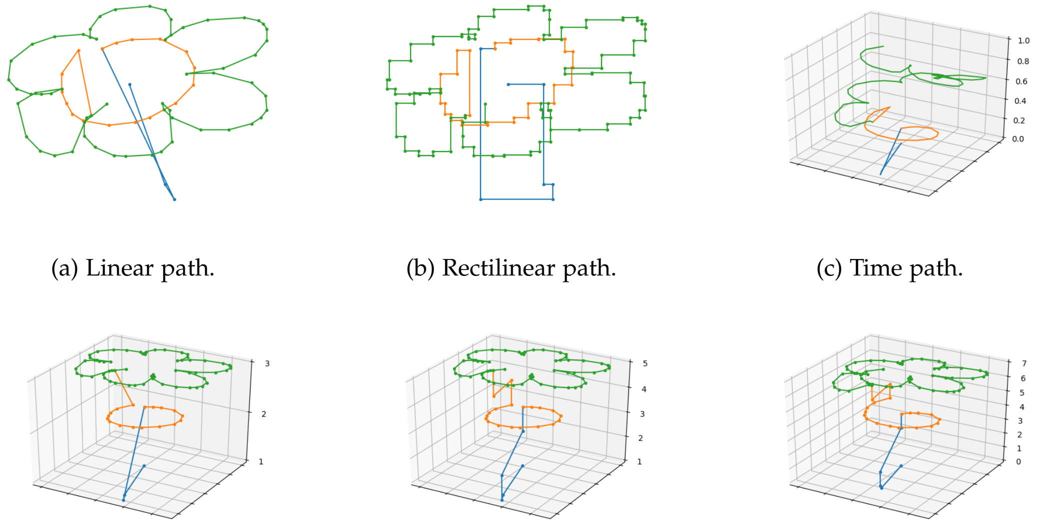
4.1 Definition and review of potential embeddings

We start in this subsection by reviewing different embeddings while adapting them to the Quick, Draw! dataset for illustrative purposes. The extension to other datasets follows immediately. All embeddings considered here are continuous piecewise linear, but their difference lies in the way this interpolation is performed. From a computational point of view, signatures of continuous piecewise linear paths can be computed with the library `iisignature`, as mentioned in Subsection 2.1. From now on, we fix a sample $\mathbf{x} \in \mathbb{R}^{3 \times p}$, which can be written as the matrix (7) where the index i has been removed to simplify notations.

Linear path. A first natural choice is to interpolate data points linearly, that is to connect each consecutive points by a straight line. Note that for the Quick Draw! data, information about pen jumps is then lost. Thus, for a particular sample, if we are given p positions of the pen $[(x_1^1, x_1^2), \ldots, (x_p^1, x_p^2)]$, we consider a partition $0 = t_1 < t_2 < \cdots < t_p = 1$ of $[0, 1]$ into p points, and define a piecewise linear path $X : [0, 1] \to \mathbb{R}^2$, which is equal to (x_j^1, x_j^2) at t_j. We end up with a two-dimensional continuous path with coordinates (X_t^1, X_t^2). This path, represented in Figure 9a, is the most often used in the literature, for example by Graham (2013), Lai et al. (2017), or Yang et al. (2016).

Rectilinear path. Another interpolation method is often used in the literature (see, e.g., Chevyrev and Kormilitzin, 2016; Kormilitzin et al., 2016) and referred to as "axis path" or "rectilinear path". It is also piecewise linear but each linear section is parallel to an axis. In other words, to move from one point (x_j^1, x_j^2) to another point (x_{j+1}^1, x_{j+1}^2), a first linear segment goes from (x_j^1, x_j^2) to (x_{j+1}^1, x_j^2), parallel to the x-axis, and a second segment from (x_{j+1}^1, x_j^2) to (x_{j+1}^1, x_{j+1}^2), parallel to the y-axis. This path is depicted in Figure 9b. A crucial aspect of this path is that there exists a simple way to reconstruct it from its signature features (Lyons and Xu, 2017). Note that for unidimensional data, such as the Urban Sound dataset, the linear and rectilinear interpolations are identical.

Time path. The third approach builds upon the linear path and enriches it by adding a monotonous coordinate. This ensures the uniqueness of the signature, as stated in Proposition 2. It usually corresponds to adding the time parametrization as a coordinate of the path, as is done by Yang et al. (2017). Therefore, if $t \mapsto (X_t^1, X_t^2)$ is the linear path described above, which



(a) Linear path.

(b) Rectilinear path.

(c) Time path.

(d) Stroke path, version 1.

(e) Stroke path, version 2.

(f) Stroke path, version 3.

Figure 9: Different embeddings of a Quick, Draw! sample. Each stroke is plotted with a different color only for the sake of illustration.

is piecewise linear, we define the time embedding as the 3-dimensional path $t \mapsto (X_t^1, X_t^2, t)$, shown in Figure 9c.

Lead-lag path. Introduced by Chevyrev and Kormilitzin (2016) and Flint et al. (2016), the lead-lag transformation has been applied by, e.g., Gyurkó et al. (2014), Kormilitzin et al. (2016), Lyons et al. (2014), and Yang et al. (2017). Building on the time path, the idea is to add lagged versions of the coordinates X^1 and X^2 as new dimensions. Let us assume that we are given p data points, and consider a partition $0 = t_1 < t_2 < \cdots < t_p < t_{p+1} = 1$ of $[0,1]$ into $p+1$ points. Then, the lead-lag path with lag 1 is defined by

$$\begin{aligned} X : [0,1] &\to \mathbb{R}^5 \\ t &\mapsto (X_t^1, X_t^2, t, X_t^3, X_t^4). \end{aligned}$$

In this definition, X^1 and X^2 are a linear interpolation of the sequence

$$\left[(x_1^1, x_1^2), \ldots, (x_p^1, x_p^2), (x_p^1, x_p^2)\right],$$

in which the last point is repeated twice, and

$$X_t^3 = \begin{cases} 0 & \text{if } t < t_1 \\ X_{t-t_1}^1 & \text{otherwise} \end{cases}, \qquad X_t^4 = \begin{cases} 0 & \text{if } t < t_1 \\ X_{t-t_1}^2 & \text{otherwise} \end{cases}. \tag{8}$$

This yields a 5-dimensional path such that the last two coordinates are delayed copies of the first two, with a delay of t_1. The process can be iterated, creating a path in $\mathbb{R}^7$ with two lags

$t \mapsto (X_t^1, X_t^2, t, X_t^3, X_t^4, X_t^5, X_t^6)$, where X^1 and X^2 are linear interpolations of the data with the last point repeated three times, X^3 and X^4 are defined by (8), and

$$X_t^5 = \begin{cases} 0 & \text{if } t < t_2 \\ X_{t-t_2}^1 & \text{otherwise} \end{cases}, \qquad X_t^6 = \begin{cases} 0 & \text{if } t < t_2 \\ X_{t-t_2}^2 & \text{otherwise} \end{cases}.$$

In this way, the lead-lag path can naturally be defined for any lag in $\mathbb{N}^*$. This path is highly dimensional (in $\mathbb{R}^7$ for a lag of 2) and cannot be represented easily. Therefore, we plot in Figure 10 some coordinates against time, namely X^1, X^3, and X^5.

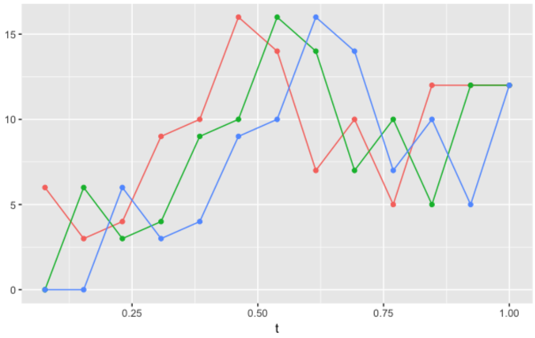


Figure 10: X_t^1 (red), X_t^3 (green), and X_t^5 (blue), coordinates of the lead-lag embedding with lag 2 against t, for $t \in [0, 1]$.

Stroke path. For the Quick, Draw! data, we are provided with extra information about pen jumps. In the context of Arabic handwriting recognition, Wilson-Nunn et al. (2018) have introduced the idea of encoding information about jumps into a new coordinate. In essence, the approach is to use a 3-dimensional path in which the last dimension corresponds to strokes, in a similar way to the encoding of matrix (7). This procedure can be deployed in various ways and we restrict our attention to three of them.

Our first approach uses the description of a drawing as given in (7). Recall that, in this matrix, the stroke categorical variable is initialized to 1, and increased by 1 each time a different stroke begins. The idea is then to simply interpolate linearly the columns of the matrix, considered as points in $\mathbb{R}^3$. As can be seen in Figure 9d, each stroke is then represented in a different horizontal plane. This procedure looks like the most natural way to encode jumps information, and from now on is called "version 1" of the stroke path.

A related approach is considered by Wilson-Nunn et al. (2018). It is represented in Figure 9e and subsequently called "version 2". Here, each stroke is indexed by odd integers, that is the first stroke is indexed by 1, the second by 3,..., and the kth by $2k - 1$. Two intermediary points are added between each stroke, indexed by even integers. For example, if $(x_{p_1}^1, x_{p_1}^2, 1)$ is the last point of the first stroke, and $(x_{p_1+1}^1, x_{p_1+1}^2, 3)$ is the first point of the second stroke, the points $(x_{p_1}^1, x_{p_1}^2, 2)$ and $(x_{p_1+1}^1, x_{p_1+1}^2, 2)$ are added to the path and linearly interpolated. This is represented in Figure 9e. The only difference with the previous path is how the path moves from one plane to another one. Instead of doing a straight line, it moves in two steps: one in a horizontal plane and another one parallel to the vertical axis. With this embedding and a recurrent network, Wilson-Nunn et al. (2018) have achieved a significant decrease in the error rate of Arabic characters recognition.

Finally, for comparison purposes, we also define a strictly monotonous coordinate. It has jumps of 1 when a new stroke begins, and otherwise grows linearly inside one stroke, such that it has increased by 1 between the beginning and the end of the stroke. In this definition, our goal is to check whether having a strictly monotonous coordinate increases accuracy, while in the two previous versions the stroke coordinate is piecewise constant. This embedding can be seen as a mix between time and stroke paths and could inherit the good properties of both. The resulting path is called "version 3" and shown in Figure 9f.

To conclude, there exists a broad range of embeddings, living in spaces of various dimensions. They lead to different signature features, which therefore do not have the same statistical properties. We will see in the next section that the embedding choice has a significant influence on accuracy results and that some strategies are more adapted than others.

4.2 Results

We present in this subsection the results of our study on embedding performance. To this end, we implement the approach depicted in Figure 3. Starting from the raw data, we first embed it into a continuous path, then compute its truncated signature, and use this vector as input for a learning algorithm. We use the embeddings described in the previous section. Note that the lead-lag path is taken with lag 1, but other lags will be discussed in Section 5.2. Each feature is normalized by the absolute value of its maximum so that all input values lie in $[-1, 1]$. We want our findings to be independent of the data and the underlying statistical model so we use a range of different algorithms. Their hyperparameters have been set to their default values, without trying to optimize them for each dataset. Indeed, we stress that our goal is not to select the best algorithm or to achieve a particularly good accuracy, but rather to compare the performance of different embeddings. The classification metric to assess prediction quality is the accuracy score. Denoting by $(y_1, \dots, y_{n_{\text{test}}})$ the test set's labels, and $(\hat{y}_1, \dots, \hat{y}_{n_{\text{test}}})$ the predicted labels, this score is defined by

$$\text{Acc}_{\text{test}} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbb{1}_{\hat{y}_i = y_i}. \tag{9}$$

The four following algorithms have been used throughout the study.

- Following Yang et al. (2017), we first consider a dense network with one hidden layer composed of 64 units with linear activation functions. We use a softmax output layer and the categorical cross-entropy loss, which yields a linear model equivalent to a logistic regression. This architecture is a sensible choice, since Proposition 3 states that linear functions of the signature approximate arbitrarily well any continuous function of the input path. We have used Python's library `keras` (Chollet et al., 2015), with TensorFlow backend. The network is regularized by adding a dropout layer after the input layer, with a rate of 0.5. Optimization is done with stochastic gradient descent with an initial learning rate of 1. It is reduced by 2 when no improvement is seen on a validation set during 10 consecutive epochs. The maximal number of epochs is set to 200 and the mini-batch size to 128.

- Furthermore, we test the performance of a random forest classifier with 50 trees, implemented in scikit-learn (Pedregosa et al., 2011). It is a nonlinear very popular method initially proposed by Breiman (2001).

- We also use the XGBoost algorithm, introduced by Chen and Guestrin (2016), and implemented in the Python package `xgboost`. It is a state-of-the-art gradient boosting technique, building upon the work of Friedman (2001). We set a maximum of 100 iterations and use early stopping with a patience of 5 to prevent overfitting and speed up training. The maximum depth of a tree is set to 3 and the minimum loss reduction to make a split to 0.5.
- Finally, we run a nearest neighbor classifier with scikit-learn implementation (Pedregosa et al., 2011), which has a default value of 5 neighbors. This method is known to suffer from the curse of dimensionality, so it will be of interest to see how the signature truncation order affects its performance.

For each of the algorithms described above and each dataset of Section 3 (Quick, Draw!, Urban Sound, and Motion Sense), the following steps are repeated:

1. Split the data into training, validation, and test sets, as described in Table 2.
2. Choose an embedding and transform samples $\mathbf{x_i}$ into continuous paths $X_i : [0,1] \to \mathbb{R}^d$.
3. For $k = 1, \dots, K$:
   (a) Compute $S^k(X_i)$, the signature truncated at order k, for every sample i. This results in training, validation and test sets of the form
   $$\{S^k(X_1), \dots, S^k(X_n)\},$$
   where $S^k(X_i) \in \mathbb{R}^{\frac{d^{k+1}-1}{d-1}}$ if $d > 1$, and $\mathbb{R}^k$ if $d = 1$.
   (b) Fit the algorithm on the training data. Validation data is used when the algorithm chosen is the linear neural network or XGboost, to adapt the learning rate and to implement early stopping, respectively.
   (c) Compute the accuracy, defined by (9), on the test set.

For a path X in $\mathbb{R}^d$, the number of features is equal to $(d^{k+1} - 1)/(d - 1)$ if $d > 1$, and to k if $d = 1$ (see Table 1 for some values). Therefore, the number of features depends on the dimension d of the embedding and the truncation order k. But d is different depending on the dataset and the embedding. Thus, to compare the quality of different embeddings, we plot the accuracy score against the log number of features, which yields one curve per embedding, where each point corresponds to a different truncation order k. We then check whether one embedding curve is above the others, which would mean that, at equal input size, this embedding is better for learning.

The results of this procedure are plotted in Figures 11, 12 and 13, which correspond respectively to the Quick, Draw!, Urban Sound and Motion Sense datasets. A first striking fact is that some embeddings, namely the time and lead-lag, seem consistently better, whatever the algorithm and the data used. It suggests that this performance is due to intrinsic theoretical properties of signatures and embeddings, not to domain-specific characteristics. It is particularly remarkable as the dimension of input streams is different from one dataset to another.

The linear and rectilinear embeddings (red and pink curves), which are often used in the literature, appear to give the worst results. These two interpolation methods do not differ much in their results, although the linear path seems to be slightly better. Moreover, it seems that the smaller the dimension d, the worse they are. Indeed, the linear embedding performs especially bad for the Urban Sound dataset, which is unidimensional, whereas the difference is

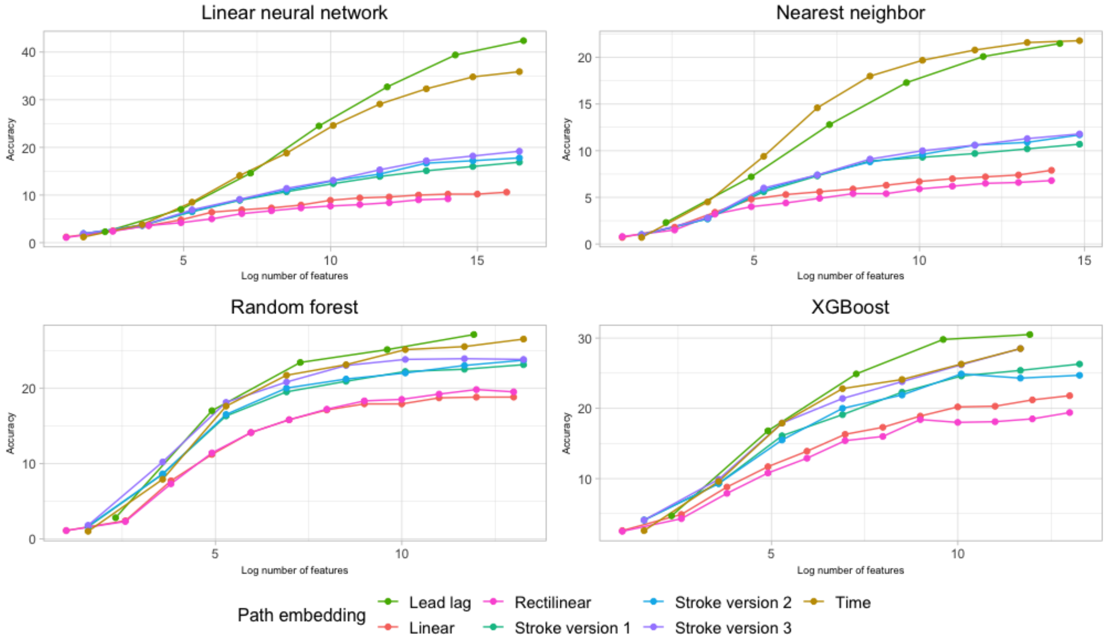


Figure 11: Quick, Draw! dataset: prediction accuracy on the test set, for different algorithms and embeddings.

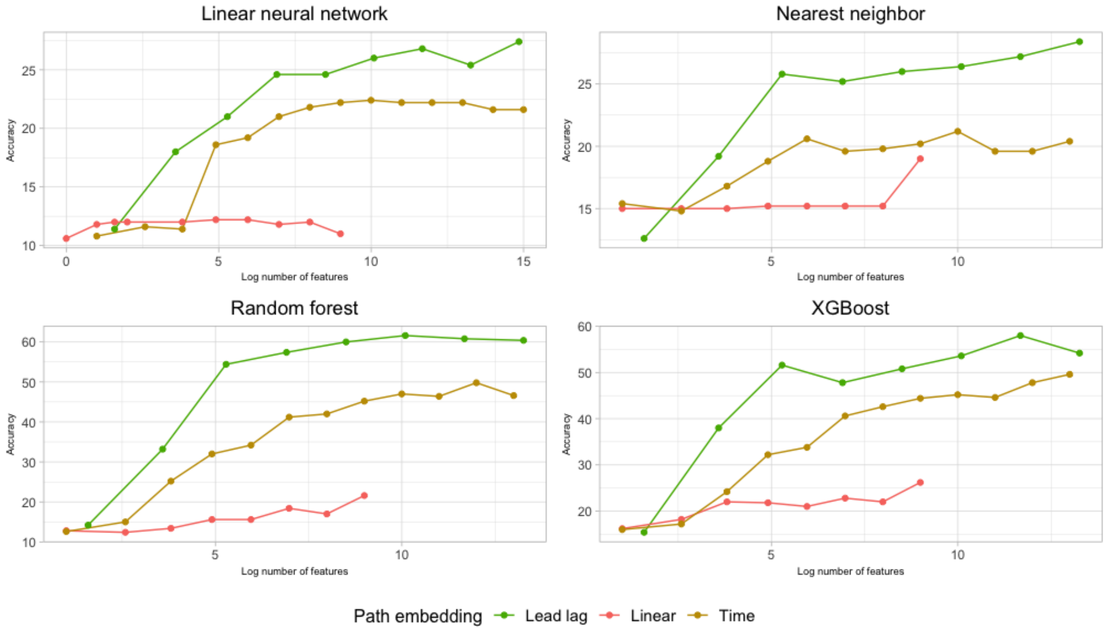


Figure 12: Urban Sound dataset: prediction accuracy for different algorithms and embeddings.

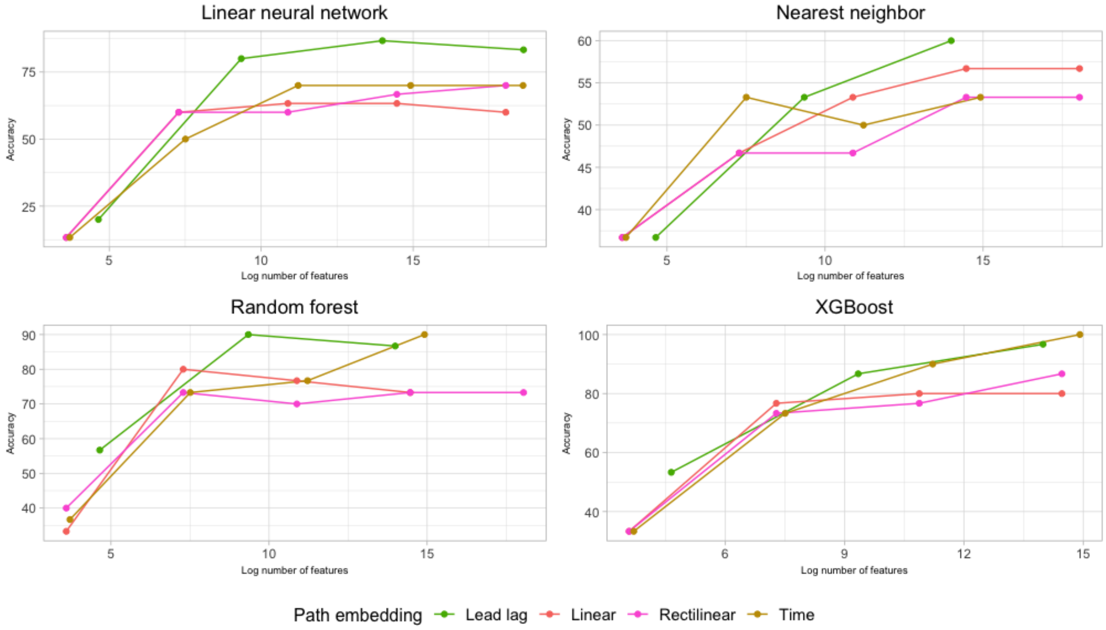


Figure 13: Motion Sense dataset: prediction accuracy on the test set, for different algorithms and embeddings.

less pronounced for the Motion Sense dataset, which has values in $\mathbb{R}^{12}$. This bad performance can be explained by the fact that there is no guarantee that the signature transformation is unique when using the linear or rectilinear embeddings. Therefore, two different paths can have the same signature, without necessarily corresponding to the same class.

On the other hand, the best embedding is the lead-lag path (green curve), followed closely by the time path (brown curve). The difference between these two embeddings is again most important for the Urban Sound dataset. For the Quick, Draw! data, stroke paths have intermediate results, better than the linear path but still worse than the time and lead-lag paths. Yet stroke paths are the only embeddings in which new information, about pen jumps, is included. It is surprising how little impact this information seems to have on prediction accuracy. Note that in all of these cases, the uniqueness of the signature is ensured so it cannot explain the performance differences.

Good performance of the lead-lag path has already been noticed in the literature. However, up to our knowledge, there are few theoretical results. Still, Flint et al. (2016) have considered a discretely sampled input path X, assumed to be a continuous semimartingale, and have studied convergence results of its associated lead-lag path, called Hoff process, when sampling frequency increases. Thus, a lot of questions remain open concerning the statistical performance of the time and lead-lag embeddings, with, to our knowledge, no theoretical result in classification or regression frameworks.

To conclude this section, the take-home message is that using the lead-lag embedding seems to be the best choice, regardless of the data and algorithm used. It does not cost anything computationally and can drastically improve prediction accuracy. Moreover, the linear and stroke paths yield surprisingly poor results, despite their frequent use in the literature.

5 Signature domain and performance

5.1 Comparison of local and global signature features

As pointed out in Section 2.2, several authors do not compute signatures on the whole time interval but use instead a partition of $[0, 1]$. The rationale for this division is to describe the path by a sequence of truncated signatures, rather than one signature computed over the whole domain. Therefore, it is not surprising that this approach is typically used in combination with recurrent neural networks. In this section, we intend to investigate if signatures computed on a sub-interval contain some local information not present in signatures of the whole path. To this end, following Wilson-Nunn et al. (2018), we consider the dyadic partition of $[0, 1]$ defined by (6):

$$0 \leq 2^{-q} < \cdots < j2^{-q} < \cdots < (2^q - 1)2^{-q} \leq 1, \quad 0 < j \leq 2^q$$

where q is the dyadic order. For different values of q, we compute signature coefficients on each interval $\big[(j-1)2^{-q}, j2^{-q}\big]$ of the dyadic partition. Therefore, for each input path X_i, we obtain a collection of signature vectors, which we concatenate altogether to obtain one big vector. This vector is then the input of a learning algorithm, and we compare the prediction accuracy curves of different dyadic orders. The time embedding with the linear neural network described in Section 4.2 is used. This process is summarized below.

1. Split the data into training, validation and test sets.

2. For $q = 0, \ldots, Q$, and $k = 1, \ldots, K$:

   (i) For $j = 1, \ldots, 2^q$, compute the signature truncated at order k on $\big[(j-1)2^{-q}, j2^{-q}\big]$, denoted by
   $$S^k(X_i)_{[(j-1)2^{-q}, j2^{-q}]},$$
   where X_i is the time embedding of sample $\mathbf{x_i}$. Repeat this over all training samples.

   (ii) For each training sample X_i, concatenate all signature vectors and obtain one vector $\tilde{S}^k(X_i)$ containing all $S^k(X_i)_{[(j-1)2^{-q}, j2^{-q}]}$, for $j = 1, \ldots, 2^q$. This yields a dataset
   $$\big\{\tilde{S}^k(X_1), \ldots, \tilde{S}^k(X_n)\big\}.$$

   (iii) Fit a linear neural network with this data as features.

   (iv) Compute accuracy on the test set.

The results of this procedure are shown in Figure 14. First, it is clear that a dyadic order of 0, which corresponds to computing the signature on the whole interval $[0, 1]$, yields the best results. Indeed, the curve is always above the others for the Quick, Draw! and Motion Sense datasets. This is less obvious for the Urban Sound dataset, as the curve $q = 0$ is really close to the one corresponding to a dyadic order of 1. However, it still achieves a better result for most truncation orders k. This difference between datasets can be linked to their length: it seems that the longer the series, the better the accuracy of thin dyadic partitions. Indeed, high dyadic orders perform best for the Urban Sound dataset, which has an average of 170 000 sampled time points (see Table 2), whereas it is clear that each new dyadic split decreases accuracy for the Quick, Draw! data, which has an average of 44 sampled points.

In a nutshell, few local information seems to be lost when the signature of the whole path is computed, although it may be worth considering partitions of the path for long streams.

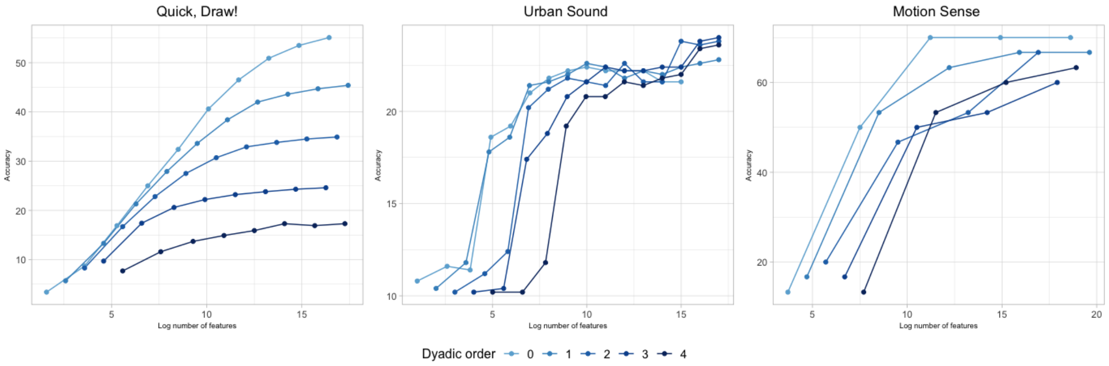


Figure 14: Test accuracy for different dyadic partitions of the path.

5.2 Performance of the signature

The message of previous sections is that the lead-lag embedding is the most appropriate in a learning context and that signatures should be computed over the whole path domain. As a natural continuation, we propose in this section to examine more closely the effect of the algorithm and to compare our prediction scores to the literature. As we will see, the signature combined with a lead-lag embedding has an excellent representation power, to the extent that it achieves prediction scores close to state-of-the-art methods, without using any domain-specific knowledge.

Before we start the comparison, we point out that the lead-lag embedding has a hyperparameter that has not yet be tuned, which is the number of lags. In our experiments, it is selected with the same approach as in previous sections: for each lag, we plot the curve of test accuracy against the number of features, where each point of the curve corresponds to one signature truncation order. We plot and select the parameter which leads to the curve above others. Figure 15 highlights that curves overlap for the Motion Sense and Urban sound cases, therefore, when there is a doubt on which curve is above, we choose to pick the smallest lag. For the Quick, Draw! and Motion Sense datasets, the best lag is then 1, whereas it is 5 for the Urban Sound dataset.

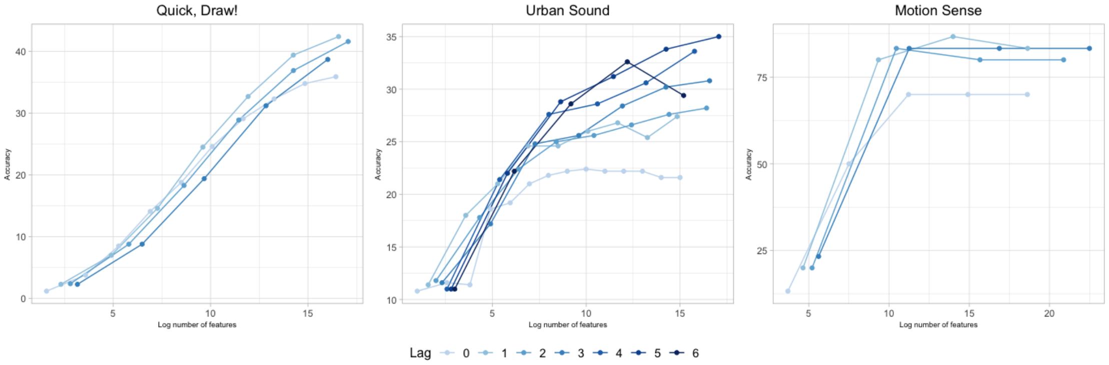


Figure 15: Test accuracy for different lags.

We also emphasize that the Motion Sense and Urban Sound datasets do not require a lot of computing resources, as they have a reasonable size (a few hundred samples for Motion Sense and thousand for Urban Sound—see Table 2) and a small number of classes. On the other hand, the Quick, Draw! recognition task, which is comprised of 340 classes, is more involved and requires an elaborate algorithm as well as a significant number of training samples. Therefore, we will use more data than in the previous sections, with 12 185 600 training samples and 87 040 validation samples.

For the Quick, Draw! dataset, we have compared our results to a Kaggle competition (Kaggle.com, 2018). In this competition, 1 316 teams competed for a prize of 25 000$. State-of-the-art deep convolutional networks, such as MobileNet or ResNet, trained with several millions of samples, were among the best competitors. Teams on the podium used ensembles of such networks. We stress that these winning methods require a lot of computing resources and are specific to images. The metric used in the competition was the mean average precision, defined as follows. Denoting by $\{y_1, \dots, y_{n_{\text{test}}}\}$ the test set labels, three ranked predictions are made for each sample, denoted by

$$\{(\hat{y}_1^1, \hat{y}_1^2, \hat{y}_1^3), \dots, (\hat{y}_{n_{\text{test}}}^1, \hat{y}_{n_{\text{test}}}^2, \hat{y}_{n_{\text{test}}}^3)\},$$

where $\hat{y}_i^1$ is the class with the largest probability, $\hat{y}_i^2$ the second largest, and so on. Then, the mean average precision at rank 3 is defined by

$$MAP_3 = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \sum_{j=1}^{3} \frac{\mathbb{1}\{\hat{y}_i^j = y_i\}}{j}.$$

Mean average precision is computed by the competition platform on 91% of a test set of 112 200 samples. We enhance the small neural network used in Section 4.2 by using ReLU activation functions and adding three hidden layers with 256 nodes. The network is trained during 300 epochs with an Adam optimizer. At each epoch, it is trained on 609 280 samples randomly selected among the 12 185 600 training samples. The best team obtains a MAP_3 of 95% whereas this small network combined with signature features truncated at order 6 already reaches 54%. The winners use an ensemble of several dozens of deep neural networks, trained on 49 million samples. This kind of architectures requires considerably more computational capacities than ours.

For the Urban Sound dataset, state-of-the-art results are obtained by Ye et al. (2017). The authors combine feature extraction with a mixture of expert models and achieve 77.36 % accuracy, defined by (9). The feature extraction step is specific to sound data and is based on several ingredients, such as whitened spectrogram, dictionary learning, soft-thresholding, recurrence quantification analysis, and so on. These crafting operations make use of a lot of domain-specific knowledge and cannot be extended easily to other applications. On the other hand, it is clear from Figure 12 that a random forest classifier performs well with signature features. Therefore, we tune its hyperparameters with a lead-lag embedding, a lag of 5, and a signature truncated at order 5. We obtain an accuracy of 70 % with 460 trees with a maximum depth of 30 and in which 500 random features are considered at each split.

Finally, Malekzadeh et al. (2019) tackle the problem of mobile sensor data anonymization. They build a deep neural network architecture that preserves user privacy but still detects the activity performed. The architecture is built on autoencoders combined with a multi-objective loss function. There is a trade-off between activity recognition and privacy but good activity recognition results are achieved. Performance of the classifier of Malekzadeh et al. (2019) is measured with the average $F1$ score, defined as follows. Assume there are L different classes, and denote by $(y_1, \dots, y_{n_{\text{test}}})$ the test labels, and by $(\hat{y}_1, \dots, \hat{y}_{n_{\text{test}}})$ the predicted ones. Then, the

$F1$ score is defined by

$$F1 = \frac{1}{L} \sum_{\ell=1}^{L} \frac{2 \cdot \mathrm{Precision}_\ell \cdot \mathrm{Recall}_\ell}{\mathrm{Precision}_\ell + \mathrm{Recall}_\ell},$$

where

$$\mathrm{Precision}_\ell = \frac{\sum_{i=1}^n \mathbb{1}\{\hat{y}_i = y_i = \ell\}}{\sum_{i=1}^n \mathbb{1}\{\hat{y}_i = \ell\}} \text{ and } \mathrm{Recall}_\ell = \frac{\sum_{i=1}^n \mathbb{1}\{\hat{y}_i = y_i = \ell\}}{\sum_{i=1}^n \mathbb{1}\{y_i = \ell\}}.$$

Malekzadeh et al. (2019) report an average $F1$ score above 92%, while the signature truncated at order 3 and combined with a XGBoost classifier achieves a $F1$ score of 93.5%. So these two scores are close, but our approach is computationally much less demanding.

We end up by stressing the fact that the algorithms used here have not been tuned to a specific application. However, the combination signature + generic algorithm achieves results close to the state-of-the-art in several domains, while requiring few computing resources and no domain-specific knowledge. Indeed, it takes approximately 52 seconds to compute the signature at order 3 of 68 000 Quick, Draw! samples on one core of a laptop, which results in 0.0008 second per sample. Besides, signature computations can be parallelized, making the approach scalable to big datasets. Lastly, the signature method achieves its best results for the high dimensional Motion Sense dataset, which suggests that it is especially relevant for multidimensional streams.

6 Conclusion

The signature method is a generic way of creating a feature set for sequential data and has recently caught the machine learning community's attention. Indeed, it yields results competitive with state-of-the-art methods, while being generic, computationally efficient, and able to handle multidimensional series. One of its appealing properties is that it captures geometric properties of the process underlying the data and does not depend on a specific basis. In this paper, we have reviewed its use in a learning process and surveyed several of its successful applications. The use of signatures relies on representing discretely sampled data as continuous paths, a mechanism called embedding. In the literature, authors use various embeddings, without any systematic comparison. We have compared different common embeddings and concluded that the lead-lag seems to be systematically better, whatever the algorithm or dataset used. Moreover, we have pointed out that the signature of the whole path appears to contain as much information as the signature of subpaths, therefore encoding both global and local properties of the input stream.

Our study is a first step towards understanding how signature features can be used in statistics, and a lot of issues remain open, both practical and theoretical. First, it would be of great interest to understand the theoretical statistical properties of embeddings, in particular, to explain the good performance of the lead-lag path. Moreover, in Section 2.2, we have seen that signature features may be combined with feedforward, recurrent, or convolutional neural networks. For each of these architectures, the point of view on signature features is different: they are considered respectively as a vector, a temporal process, or an image. A more detailed understanding of these representations would be valuable. Finally, it could be worth investigating the robustness of the signature method when the truncation order becomes large. Indeed, Figures 11, 13, and 12 suggest that the signature is robust to dimension: the accuracy curves do not decrease when the number of features becomes large, even when a nearest neighbor algorithm is used with more than a hundred thousand features. In this situation, we would expect the variance to be large and to reduce accuracy, but that does not seem to be the case. This phenomenon may deserve a more in-depth study.

References

A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31:606–660, 2017.

A. Bagnall, J. Lines, W. Vickers, and E. Keogh. The uea & ucr time series classification repository. 2018. Available at: www.timeseriesclassification.com.

D. J. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. In *Advances in knowledge discovery and data mining*, pages 229–248. AAAI/MIT Press, Menlo Park, 1996.

G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time Series Analysis: Forecasting and Control. 5th Edition.* Wiley, Hoboken, 2015.

L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

J. Chang, N. Duffield, H. Ni, and W. Xu. Signature inversion for monotone paths. *Electronic Communications in Probability*, 22:1–11, 2017.

K.-s. Chen. Integration of paths—a faithful representation of paths by non-commutative formal power series. *Transactions of the American Mathematical Society*, 89:395–407, 1958.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, New York, 2016. ACM.

I. Chevyrev and A. Kormilitzin. A primer on the signature method in machine learning. *arXiv:1603.03788*, 2016.

F. Chollet et al. Keras, 2015. Available at: https://github.com/keras-team/keras.

H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33:917–963, 2019.

F. Ferraty and P. Vieu. *Nonparametric Functional Data Analysis: Theory and Practice*. Springer, New York, 2006.

G. Flint, B. Hambly, and T. Lyons. Discretely sampled signals and the rough Hoff process. *Stochastic Processes and their Applications*, 126:2593–2614, 2016.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29:1189–1232, 2001.

P. K. Friz and N. B. Victoir. *Multidimensional Stochastic Processes as Rough Paths: Theory and Applications*, volume 120 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 2010.

Google. The quick, draw! dataset, 2017. "Data made available by Google, Inc. under the Creative Commons Attribution 4.0 International license. Available from: https://github.com/creativelab/quickdraw-dataset,".

B. Graham. Sparse arrays of signatures for online character recognition. *arXiv:1308.0371*, 2013.

L. G. Gyurkó, T. Lyons, M. Kontkowski, and J. Field. Extracting information from the signature of a financial data stream. *arXiv:1307.7244*, 2014.

M. Hairer. Solving the KPZ equation. *Annals of Mathematics*, 178:559–664, 2013.

B. Hambly and T. Lyons. Uniqueness for the signature of a path of bounded variation and the reduced path group. *Annals of Mathematics*, 171:109–167, 2010.

J. D. Hamilton. *Time Series Analysis*. Princeton University Press, Princeton, 1994.

Kaggle.com. Quick, draw! doodle recognition challenge, 2018. Available from: https://www.kaggle.com/c/quickdraw-doodle-recognition/overview.

F. J. Király and H. Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research*, 20:1–45, 2019.

A. Kormilitzin, K. Saunders, P. Harrison, J. Geddes, and T. Lyons. Application of the signature method to pattern recognition in the cequel clinical trial. *arXiv:1606.02074*, 2016.

S. Lai, L. Jin, and W. Yang. Online signature verification using recurrent neural network and length-normalized path signature descriptor. In *Proceedings of the 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 400–405. IEEE, 2017.

M. Liu, L. Jin, and Z. Xie. Ps-lstm: Capturing essential sequential online information with path signature and lstm for writer identification. In *Proceedings of the 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 664–669. IEEE, 2017.

H. Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, Berlin, 2005.

T. Lyons and W. Xu. Inverting the signature of a path. *Journal of the European Mathematical Society*, 20:1655–1687, 2018.

T. Lyons, H. Ni, and H. Oberhauser. A feature set for streams and an application to high-frequency financial tick data. In *Proceedings of the 2014 International Conference on Big Data Science and Computing*, page 5. ACM, 2014.

T. J. Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 14:215–310, 1998.

T. J. Lyons and W. Xu. Hyperbolic development and inversion of signature. *Journal of Functional Analysis*, 272:2933–2955, 2017.

T. J. Lyons, M. Caruana, and T. Lévy. *Differential Equations driven by Rough Paths*, volume 1908 of *Lecture Notes in Mathematics*. Springer, Berlin, 2007.

M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi. Protecting sensory data against sensitive inferences. In *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*. ACM, New-York, 2018.

M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi. Mobile sensor data anonymization. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, pages 49–58. ACM, New-York, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. O. Ramsay and B. W. Silverman. *Functional Data Analysis. 2nd Edition.* Springer, New York, 2005.

J. Reizenstein and B. Graham. The iisignature library: efficient calculation of iterated-integral signatures and log signatures. *arXiv:1802.08252*, 2018.

J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044. ACM, 2014.

D. Wilson-Nunn, T. Lyons, A. Papavasiliou, and H. Ni. A path signature approach to online arabic handwriting recognition. In *Proceedings of the IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR)*, pages 135–139. IEEE, 2018.

W. Yang, L. Jin, and M. Liu. Chinese character-level writer identification using path signature feature, dropstroke and deep cnn. In *Proceedings of the 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 546–550. IEEE, 2015.

W. Yang, L. Jin, and M. Liu. Deepwriterid: An end-to-end online text-independent writer identification system. *IEEE Intelligent Systems*, 31:45–53, 2016.

W. Yang, T. Lyons, H. Ni, C. Schmid, L. Jin, and J. Chang. Leveraging the path signature for skeleton-based human action recognition. *arXiv:1707.03993*, 2017.

J. Ye, T. Kobayashi, and M. Murakawa. Urban sound event classification based on local and global features aggregation. *Applied Acoustics*, 117:246–256, 2017.