SIGNATURES FOR HAND KINEMATIC SYNERGIES

ALAN CHEN AND ALEXEY IZMAILOV

To our dearest MCM teammate, Matthew Meeker

1. Introduction

Patterns underlying hand grasp motions lie at the crux of research into the development of assistive devices and prostheses. However, this complex coordination task presents a high-dimensional kinematic problem in which several features of the human hand interact in poorly understood ways (joints alone have 27 degrees of freedom). Several past studies have implicitly reduced dimensionality by focusing on hand postures [SFS98] rather than entire grasping motions while others have attempted direct computational methods [Jar+19] such as Principal Component Analysis. As an alternative dimensionality reduction technique, we study the signature method in combination with classical machine learning models.

Recent developments in rough path theory have resulted in increased attention to path signatures in the mathematical community. One of the advantages of a signature-based approach lies in the ability of this computational object to capture key geometric and analytic properties of sequential data. Past successes of these methods have included online character recognition [YJL15] and time-series analysis and regression in a financial setting [LLN16]. Moreover, in combination with standard techniques from machine learning, signature methods have resulted in successful classification tasks in learning behavioral patterns of patients with bipolar disorder [Arr+18]. Our aim is to improve upon the methodologies used in these studies in the context of classifying kinematic hand synergies.

2. Theory

2.1. **Notation.** $[k]$ denotes $\{1, 2, \ldots, k\}$. $X^{(I)}$ for an index set $I = \{i_1, \ldots, i_n\}$ denotes the n dimensional path $X^{(I)} = \{X_{i_1}, \ldots, X_{i_n}\}$ (pairs (i_j, i_k) for $j \neq k$ need not be distinct).

2.2. **Continuous Paths from Raw Sequential Data.** Consider a collection of N d-tuples $\{X_{1_i}, X_{2_i}, \ldots, X_{d_i}\}_{i=1}^{N}$, where each X_{k_i} represents a single one-dimensional datum. This set of data can be embedded into a continuous path via interpolation. In this series of experiments, we exclusively use linear interpolation but any method is usable given a result such as Theorem 2.1.

2.3. **Path Signature.** The signature of a path is defined as a sequence of coordinate iterated integrals. Let $X : [0, T] \to \mathbb{R}^d$ be a continuous path with finite p-variation for some $p < 2$. Then for a multi-index $I = (i_1, i_2, \ldots, i_k)$, the iterated integrals of order k with this multi-index are defined as

$$S(X)^I = \int_{0<u_1<u_2<\cdots<u_k<T} dX_{u_2}^{i_1} dX_{u_2}^{i_2} \cdots dX_{u_k}^{i_k} \tag{2.1}$$

which exists because of the finite p variation condition, and the signature $S(X)$ of X is the infinite sequence
(2.2)
$$S(X) = \left(1, S(X)^{(1)}, S(X)^{(2)}, \cdots, S(X)^{(d)}, S(X)^{(1,1)}, S(X)^{(1,2)}, \cdots\right) \in \prod_{i=0}^{\infty} (\mathbb{R}^d)^{\otimes i}$$

with the multi-index I consisting of all combinations of $i_j \in \{1, 2, \ldots, d\}$ for $j = 1, 2, \ldots, k$. $S(X)$ is in the dual space $T(\mathbb{R}^d)^*$ as mentioned, so that one can think of the multi-indices as elements of $T(\mathbb{R}^d)$, and we take the corresponding coordinate $S(X)^I$ when applying $S(X)$. We investigate this deeper in later sections.

The signature can also be treated as a formal power series with coordinate-wise addition, scalar multiplication, and the tensor product.

There are various useful gimmicks that come from the nice properties and definition of the signature of smooth paths. For example, for certain classes of paths, the signature can be directly computed using an explicit formula. As an example we directly compute the signature for the class of linear paths.

Lemma 2.1 (Signature of Line)**.** *Let $X : [0, T] \to \mathbb{R}^d$ be linear in all dimensions. For a multi-index $I = (i_1, i_2, \ldots, i_k)$, the signature of X on $[0, T]$ can be computed as*

$$S(X)^I = \frac{1}{k!} \prod_{j=1}^{k} \left(X^{(i_j)}(T) - X^{(i_j)}(0)\right). \tag{2.3}$$

Proof. Because X is linear in all dimensions,

$$\frac{dX^{(i)}}{dt} = \frac{X^{(i)}(T) - X^{(i)}(0)}{T} \quad \forall i \in [d]. \tag{2.4}$$

Computing the signature using Equation 2.1 and Equation 2.4,

$$\begin{aligned}
S(X)^I &= \int_{0<u_1<u_2<\cdots<u_k<T} dX^{i_1}_{u_2} dX^{i_2}_{u_2} \cdots dX^{i_k}_{u_k} \\
&= \frac{1}{T^k} \left(\prod_{j=1}^{k} X^{(i_j)}(T) - X^{(i_j)}(0)\right) \int_{0<u_1<u_2<\cdots<u_k<T} du_1 du_2 \cdots du_k \\
&= \frac{1}{T^k} \left(\prod_{j=1}^{k} X^{(i_j)}(T) - X^{(i_j)}(0)\right) \left(\frac{T^k}{k!}\right) \\
&= \frac{1}{k!} \prod_{j=1}^{k} X^{(i_j)}(T) - X^{(i_j)}(0).
\end{aligned}$$

Clearly, linearity in all dimensions is not necessary; it is only needed in the k dimensions I of interest. The nice paths encountered in this report will indeed be linear in all dimensions due to linear interpolation. □

We now turn to some important properties of the signature.

2.4. Time Reparameterizations.

A surjective, smooth, and nondecreasing mapping $\psi : [0, T] \to [0, T]$ is called a **time reparameterization**. The signature is invariant under time reparameterizations.

Theorem 2.2 (Invariance Under Time Reparameterizations)**.** *Let $X : [0, T] \to \mathbb{R}^d$ be a path. Let $\tilde{X} : [0, T] \to \mathbb{R}^d$ be defined as*

$$\tilde{X}(t) = X(\psi(t)).$$

Then, if $I = (i_1, i_2, \ldots, i_k)$,

$$S(\tilde{X})^I = S(X)^I. \tag{2.5}$$

Proof. First, this can be shown for any general path integral. Let $X, Y : [0, T] \to \mathbb{R}$ be two one dimensional paths, and $\tilde{X}$ and $\tilde{Y}$ be their respective reparameterizations which still have nice properties because ψ is assumed to be nice. Then,

$$\int_0^T \tilde{Y}(t) d\tilde{X}(t) = \int_0^T \tilde{Y}(t) \frac{dX(\psi(t))}{d\psi} \frac{d\psi}{dt} dt. \tag{2.6}$$

Using a simple change of variables $u = \psi(t)$ and $du = \frac{d\psi}{dt} dt$,

$$= \int_0^T Y(u) \frac{dX}{du} du = \int_0^T Y(u) dX, \tag{2.7}$$

which as expected.

Now, noting that the signature as defined in Equation 2.1 is just k path integrals, it becomes apparent that this argument can be applied identically k times to conclude the invariance of signatures under time reparameterizations. □

Intuitively, this property should be interpreted that the signature's values do not care about how *quickly* an action is performed; they only depend on the *shape* of the path. A formal generalization of this intuition to beyond just smooth paths is "tree-like" equivalence [HK14].

2.5. **Important Identities.** A very interesting property of signatures is their relation to linearity. Namely, there is an interesting identity that relates products of lower level signature terms to a linear combination of higher level signature terms.

2.5.1. *Shuffle Algebra.* First, define an algebra $\mathcal{U}$ over a commutative associative ring with identity element e. Select $d \in \mathbb{Z}^+$ and define $\mathcal{U}$ with the basis

$$\{e_\emptyset\} \cup \{e_{(i_1, \ldots i_n)} : i_k \in [d] \forall k \in [n] \text{ and } n = 1, 2, \ldots\}. \tag{2.8}$$

e is also the identity element in $\mathcal{U}$ with respect to the multiplication $\sqcup\!\sqcup : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ that we will define now. In order to define $\sqcup\!\sqcup$, we define the bilinear "append" operation $\vee : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ as

$$e_{(i_1, i_2, \ldots, i_k)} \vee e_{(i_{k+1})} = e_{(i_1, i_2, \ldots, i_k, i_{k+1})}. \tag{2.9}$$

Now, we can define the shuffle product $\sqcup\!\sqcup : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ recursively for two finite **ordered** index sets $I = (i_1, i_2, \ldots, i_k)$ and $J = (j_1, j_2, \ldots, j_m)$.

$$\begin{aligned} e_{(i_1, \ldots, i_k)} \sqcup\!\sqcup\, e_{(j_1, \ldots j_m)} =& [e_{(i_1, \ldots, i_{k-1})} \sqcup\!\sqcup\, e_{(j_1, \ldots j_m)}] \vee e_{i_k} \\ &+ [e_{(i_1, \ldots, i_k)} \sqcup\!\sqcup\, e_{(j_1, \ldots j_{m-1})}] \vee e_{i_m}, \end{aligned} \tag{2.10}$$

and $e_\emptyset \sqcup\!\sqcup\, e_I = e_I \sqcup\!\sqcup\, e_\emptyset = e_I$. It can be shown that $(\mathcal{U}, \sqcup\!\sqcup)$ is a unital commutative associative algebra.

2.5.2. *Signatures Shuffle Algebra.* Under this shuffle algebra framework, we can define $\mathcal{U} = \bigoplus_{i=0}^{\infty}(\mathbb{R}^d)^{\otimes i}$, so that we can define the following *signature mapping.*

Definition 2.3 (Signature Mapping)**.** The **signature mapping** $\langle \cdot, \cdot \rangle$ is defined as

$$\langle \cdot, \cdot \rangle : \prod_{i=0}^{\infty}(\mathbb{R}^d)^{\otimes i} \times \bigoplus_{i=0}^{\infty}(\mathbb{R}^d)^{\otimes i} \to \mathbb{R}, \quad (S(X), e_I) \mapsto S(X)^I. \tag{2.11}$$

Observe that $\langle \cdot, \cdot \rangle$ is linear because it is just path integrals. [Ree58] noted the following **shuffle product identity** holds:

Theorem 2.4 (Shuffle Product Identity)**.** *Let $I = (i_1, i_2, \dots, i_k)$ and $J = (j_1, j_2, \dots, j_m)$ be two index sets and $X : [0, T] \to \mathbb{R}^d$ be so that $S(X)$ is well defined. Then,*

$$\langle S(X), e_I \sqcup\!\sqcup\, e_J \rangle = \langle S(X), e_I \rangle \langle S(X), e_J \rangle. \tag{2.12}$$

Proof. The shuffle product identity can be shown through induction on $|I| + |J|$. The base cases of $|I| + |J| < 2$ are straightforward via direct computation. Because $|I|, |J| \in \mathbb{Z}^+ \cup \{0\}$, WLOG there are 2 subcases.

Subcase 1: $|I| = |J| = 0$. Then, by definition,

$$\langle S(X), e_I \rangle \langle S(X), e_J \rangle = 1^2 = 1 = \langle S(X), e_\emptyset \rangle = \langle S(X), e_I \sqcup\!\sqcup\, e_\emptyset \rangle = \langle S(X), e_I \sqcup\!\sqcup\, e_J \rangle.$$

Subcase 2: $|I| = 1$, $|J| = 0$. Then,

$$\langle S(X), e_I \rangle \langle S(X), e_J \rangle = \langle S(X), e_I \rangle (1) = \langle S(X), e_I \sqcup\!\sqcup\, e_\emptyset \rangle = \langle S(X), e_I \sqcup\!\sqcup\, e_J \rangle.$$

So, as the inductive hypothesis assume that the shuffle product identity is true for all I', J' such that $|I'| + |J'| < k + m$. We want to show that it is also true for our current index sets I, J where $|I| + |J| = k + m$.

For the sake of argument, define a "parameterized signature" mapping $S_t : \mathcal{U} \to \mathbb{R}$ as follows:

$$S_t(e_{(i_1, \dots, i_k)}) = \int_0^t S_u(e_{(i_1, \dots, i_{k-1})}) dX^{i_k}(u), \tag{2.13}$$

with the base case that $S_t(e_\emptyset) = 1$. For intuition, observe that $\langle S(X), e_I \rangle = S(X)^I = S_T(e_I)$.

We note the interplay between this definition of the signature with the operation $\vee$. Namely, for any $e_I \in \mathcal{U}$,

$$S_t(e_I \vee e_i) = \int_0^t S_u(e_I) dX^i(u). \tag{2.14}$$

In particular, we want to know more about the product $S_t(e_I)S_t(e_J)$. Apply integration by parts and the inductive definition of the signature to see that

$$\begin{aligned} S_t(e_I)S_t(e_J) &= \int_0^t S_u(e_I) dS_u(e_J) + \int_0^t S_u(e_J) dS_u(e_I) \\ &= \int_0^t S_u(e_I) S_u(e_{(j_1, \dots j_{m-1})}) dX^{j_m}(u) \\ &\quad + \int_0^t S_u(e_J) S_u(e_{(i_1, \dots i_{k-1})} dX^{i_k}(u). \end{aligned}$$

Note the level of both integrands:

$$\begin{aligned} |I| + |\{j_1, \dots j_{m-1}\}| &= k + (m-1) < k + m, \\ |J| + |\{i_1, \dots, i_{k-1}\}| &= m + (k-1) < k + m. \end{aligned}$$

Thus, applying the inductive hypothesis reveals that

$$\begin{aligned} S_t(e_I)S_t(e_J) &= S_t((e_{(i_1,\dots,i_k)} \sqcup\!\sqcup\, e_{(j_1,\dots,j_{m-1})}) \vee e_{j_m}) \\ &\quad + S_t((e_{(i_1,\dots,i_{k-1})} \sqcup\!\sqcup\, e_{(j_1,\dots,j_m)}) \vee e_{i_{k-1}}) \\ &= S_t((e_{(i_1,\dots,i_k)} \sqcup\!\sqcup\, e_{(j_1,\dots,j_{m-1})}) \vee e_{j_m} \\ &\quad + (e_{(i_1,\dots,i_{k-1})} \sqcup\!\sqcup\, e_{(j_1,\dots,j_m)}) \vee e_{i_{k-1}}) \\ &= S_t(e_I \sqcup\!\sqcup\, e_J). \end{aligned}$$

Now just let $t = T$, and the desired result follows from induction. □

Corollary 2.4.1. *Any polynomial of signature terms can be expressed as a linear combination of other signature terms.*

Proof. This inductively follows from the shuffle product identity, as any polynomial term of signature terms of degree d can be written as a sum of polynomial signature terms of degree $d-1$. We can continue applications of the shuffle product identity on all polynomial terms until $d = 2$, wherein applying the shuffle product identity reduces all terms to degree $2 - 1 = 1$ i.e. a linear combination. □

2.5.3. *Chen's Identity.* Consider two paths $X : [0, T_1] \to \mathbb{R}^d$ and $Y : [0, T_2] \to \mathbb{R}^d$. Define $Z : [0, T_1 + T_2] \to \mathbb{R}^d$ as the **concatenation** of X and Y:

$$Z(t) = \begin{cases} X(t) & t \in [0, T_1] \\ X(T_1) + (Y(t - T_1) - Y(0)) & t \in (T_1, T_1 + T_2]. \end{cases} \tag{2.15}$$

The following result is incredibly useful for computing signatures of a concatenated path, as it reduces a tricky integral to the tensor product of two simpler signatures.

Theorem 2.5 (Chen's Identity)**.** *For any two paths $X : [0, T_1] \to \mathbb{R}^d$, $Y : [0, T_2] \to \mathbb{R}^d$ and their concatenation $Z : [0, T_1 + T_2] \to \mathbb{R}^d$,*

$$S(Z) = S(X) \otimes S(Y). \tag{2.16}$$

More specifically, for any index set $I = (i_1, i_2, \dots, i_k)$,

$$S(Z)^I = \sum_{j=0}^{k} S(X)^{(i_1,\dots,i_j)} S(Y)^{(i_{j+1},\dots,i_k)}. \tag{2.17}$$

Proof. To prove Chen's relationship, we can use direct computation.

If $I = \emptyset$, then in both Equation 2.16 and Equation 2.17, both sides evaluate to 1, so the relationship holds.

Now, let I_k represent a nonempty index set. Setting up the signature computation,

$$S(Z)^{I_k} = \int_{0 < t_1 < \cdots < t_k < T_1 + T_2} dZ_{(I_k)}. \tag{2.18}$$

This computation splits up into the sum of integrals on sets of this form:

$$A_j = \{[t_i]_{i=1}^k : 0 < t_1 < \cdots < t_j \le T_1 < t_{j+1} < \cdots < t_k < T_1 + T_2\}.$$

for all j. First, we prove that $A_{j_1} \cap A_{j_2} = \emptyset$ if $j_1 \ne j_2$. WLOG, let $j_1 < j_2$. Notice that

$$A_{j_1} \subseteq \{[t_i]_{i=1}^k k : T_1 < t_{j_1+1} < T_1 + T_2\}$$

but

$$A_{j_2} \subseteq \{[t_i]_{i=1}^k k : 0 < t_{j_1+1} \le T_1\}.$$

Because the sets on the RHS have no overlap, it follows that $A_{j_1} \cap A_{j_2} = \emptyset$ too. Thus, since

$$\bigcup_{j=0}^{k} A_j = \{[t_i]_{i=1}^k : 0 < t_1 < \cdots < t_k < T_1 + T_2\},$$

the collection $\{A_j\}_{j=0}^k$ is in fact a partition of the original set. Specifically,

$$S(Z)^{I_{k+1}} = \sum_{j=0}^{k} \int_{A_j} dZ_{(I_{k+1})}. \tag{2.19}$$

We can split up this integral with respect to A_j again into a product of two lower integrals because Z is defined as a concatenation. These integrals are defined on the following sets:

$$\begin{aligned} A_{j,1} &= \{0 < t_1 < \cdots < t_j < T_1\}, \\ A_{j,2} &= \{T_1 < t_{j+1} < \cdots < t_k < T_1 + T_2\}. \end{aligned}$$

Thus, Equation 2.19 becomes

$$\begin{aligned} &= \sum_{j=0}^{k} \int_{A_{j,1}} \int_{A_{j,2}} dZ_{(i_{j+1},\ldots,i_k)} dX_{(i_1,\ldots,i_j)} \\ &= \sum_{j=0}^{k} \int_{A_{j,1}} dX_{(i_1,\ldots,i_j)} \int_{A_{j,2}} dZ_{(i_{j+1},\ldots,i_k)}. \end{aligned} \tag{2.20}$$

These terms look awfully familiar. Reparameterizing $A_{j,2}$ so that everything is shifted left by T_1, these integrals are exactly the signature terms. Formally,

$$A_{j,2} = \{0 < t_{j+1} - T_1 < \cdots < t_k - T_1 < T_2\}.$$

Since the ts are just dummy variables, Equation 2.20 just works out to be

$$\sum_{j=0}^{k} S(X)^{(i_1,\ldots,i_j)} S(Y)^{(i_{j+1},\ldots i_k)}$$

as desired. □

One can observe similarities in structure to some important definitions from the shuffle product identity, namely the deconcatenation operator Δ.

2.6. **Computing Signatures.**

Now, consider the problem of computing the signature of a discrete data stream such as in subsection 2.2. The data stream as a continuous path through an interpolation method. Alternative techniques such as Monte Carlo integration can be attempted, but because of Chen's identity there exists an efficient and analytically exact method. Computing a signature can be broken down into three steps:

(1) Interpolate data points
(2) Compute signatures on interpolated segments
(3) Concatenate interpolated segments with Chen's identity

Step (2) is usually easy: assuming a "nice" and uniformly applied interpolation method, the signature can be analytically precomputed. For example, for linear interpolation, Theorem 2.1 can be applied.

Assuming linear interpolation, we can analyze the full algorithmic complexity of computing $S(X)^I$ for some I with $|I| \leq d$. Step (1) is $O(N)$, step (2) is $O(Nd)$, and finally step (3) is $O(d)$ as well. Thus, the signature can be computed in $O(Nd)$ time, which is fast enough for almost all applications, where d is usually $O(10^2)$ or smaller.

This computation is implemented under the hood in the `iisignature` Python package [RG20].

2.7. Log-Signature.

Continue to consider the shuffle algebra $(\mathcal{U}, \sqcup\!\sqcup)$ as defined in subsubsection 2.5.1. If the d dimensional path takes values in $\mathbb{R}^d$, $\mathcal{U}$ can be thought of as $T(\mathbb{R}^d) = \bigoplus_{i=1}^{\infty} (\mathbb{R}^d)^{\otimes i}$, or the tensor space over $\mathbb{R}^d$.

Now, consider the dual space $T(\mathbb{R}^d)^*$ and a particular subset of functionals which are defined to be the **characters** (equiv. algebraic homomorphisms). A functional g is defined to be a character if it respects the shuffle product i.e.

$$g(X \sqcup\!\sqcup Y) = g(X) \cdot g(Y) \tag{2.21}$$

where $\cdot$ is multiplication in $\mathbb{R}$. Notice, necessarily, $g(e_\emptyset) = 1$ or 0. However, we choose not to include $g(e_\emptyset) = 0$ since that would be an noninvertible element as we will see later. Only one function is being thrown out - if $g(e_\emptyset)) = 0$, then $g(e_I) \equiv 0$ for all e_I.

We also define the **deconcatenation** operator $\Delta : T(\mathbb{R}^d) \to T(\mathbb{R}^d) \otimes T(\mathbb{R}^d)$ as

$$\Delta[e_{(i_1,\ldots,i_n)}] = \sum_{j=0}^{n} e_{(i_1,\ldots,i_j)} \otimes e_{(i_{j+1},\ldots,i_n)}. \tag{2.22}$$

It turns out that the set of all characters and the dual of the deconcatenation operator (appropriately called the concatenation operator) $* : T(\mathbb{R}^d)^* \times T(\mathbb{R}^d)^* \to T(\mathbb{R}^d)^*$:

$$(g * h)(e_I) = (g \tilde{\otimes} h)(\Delta e_I) = \sum_{j=0}^{n} g(e_{(i_1,\ldots,i_j)}) h(e_{(i_{j+1},\ldots,i_n)}). \tag{2.23}$$

form a group (Characters, $*$), where $g\tilde{\otimes}h$ is just a notation to reveal the relationship to the deconcatenation operator. $g\tilde{\otimes}h$ means "evaluate g on the left and h on the right."

Lemma 2.6 (Character Group)**.** *(Characters, $*$) forms a group.*

Proof. We need to show associativity, existence of identity, and existence of inverses in Characters with respect to $*$.

Firstly, associativity is easy from definition, since the $*$ operation decomposes down into associative operations in the underlying $\mathbb{R}^d$.

We want to show for any $f, g, h \in T(\mathbb{R}^d)^*$,

$$f * (g * h) = (f * g) * h. \tag{2.24}$$

Expanding from definition,

$$\begin{aligned}f*(g*h)(e_I) &= \sum_{j=0}^{n} f(e_{(i_1,\dots,i_j)})(g*h)(e_{(i_{j+1},\dots,i_n)})\\ &= \sum_{j=0}^{n} f(e_{(i_1,\dots,i_j)}) \sum_{k=j}^{n} g(e_{(i_{j+1},\dots,i_k)})h(e_{(i_{k+1},\dots,i_n)})\\ &= \sum_{j=0}^{n}\sum_{k=j}^{n} f(e_{(i_1,\dots,i_j)})g(e_{(i_{j+1},\dots,i_k)})h(e_{(i_{k+1},\dots,i_n)}).\end{aligned}$$

Since these are finite sums we can exchange the order:

$$\begin{aligned}&= \sum_{k=0}^{n}\sum_{j=0}^{k} f(e_{(i_1,\dots,i_j)})g(e_{(i_{j+1},\dots,i_k)})h(e_{(i_{k+1},\dots,i_n)})\\ &= \sum_{k=0}^{n}\left(\sum_{j=0}^{k} f(e_{(i_1,\dots,i_j)})g(e_{(i_{j+1},\dots,i_k)})\right) h(e_{(i_{k+1},\dots,i_n)})\\ &= \sum_{k=0}^{n}(f*g)(e_{(i_1,\dots,i_k)})h(e_{(i_{k+1},\dots,i_n)})\\ &= (f*g)*h.\end{aligned}$$

Next, the identity element is just $e \in T(\mathbb{R}^d)^*$ where

$$e(e_I) = \begin{cases} 1 & I = \emptyset \\ 0 & \text{otherwise.}\end{cases} \tag{2.25}$$

This is a character because it maps any nontrivial set to 0, but also $e(f(\emptyset)\sqcup f(\emptyset)) = 1$ and $e(f(\emptyset))e(f(\emptyset)) = 1$. Clearly, for any $g \in T(\mathbb{R}^d)$ and $e_I \in T(\mathbb{R}^d)$, by definition in Equation 2.23,

$$(g*e)(e_I) = (e*g)(e_I) = g(e_I). \tag{2.26}$$

The inverse of any $g \in$ Characters can be derived inductively, since by definition we want $(g*g^{-1})(e_I) = 0$ for all $e_I \in \mathcal{U}\setminus\{e_\emptyset\}$. It can be verified that the following definition is indeed a correct inverse in the character group through an inductive argument as well.

Define the set of ordered k-partitions of an index set of size n as

$$\Pi_{k,n} = \left\{ \{x_1,\dots x_k\} : \sum_{i=1}^{k} x_i = n, x_i > 0 \text{ for all } i \in [k] \right\}. \tag{2.27}$$

For all $\pi \in \Pi_{k,n}$, define π_m for $m \in [k]$ as

$$\pi_m = \left(i_{1+\sum_{j=1}^{m-1} x_j}, \dots, i_{\sum_{j=1}^{m} x_j} \right). \tag{2.28}$$

Then, for $I = \{i_1,\dots,i_n\}$, it can be shown that

$$g^{-1}(e_I) = \sum_{k=1}^{n}\sum_{\pi\in\Pi_{k,n}} (-1)^k \prod_{j=1}^{k} g(e_{\pi_j}) \tag{2.29}$$

is the unique inverse for g. To see how, again we can work from definition - consider e_I where $|I| \geq 1$ so that $I = \{i_1, \dots, i_n\}$.

$$\begin{aligned}(g * g^{-1})(e_I) &= \sum_{j=0}^{n} g(e_{(i_1,\dots,i_j)})g^{-1}(e_{(i_{j+1},\dots,i_n)}) \\ &= \underbrace{\sum_{k=1}^{n} \sum_{\pi \in \Pi_{k,n}} (-1)^k \prod_{j=1}^{k} g(e_{\pi_j})}_{\text{Term 1}} + \underbrace{\sum_{j=1}^{n} g(e_{(i_1,\dots,i_j)})g^{-1}(e_{(i_{j+1},\dots,i_n)})}_{\text{Term 2}} .\end{aligned}$$

Consider some arbitrary term in Term 1. It looks like

$$(-1)^k \prod_{j=1}^{k} g(e_{\pi_j})$$

for some $\pi \in \Pi_{k,n}$. Notice that a mirrored term appears in Term 2 that looks like

$$g(e_{\pi_1}) \left((-1)^{k-1} \prod_{j=2}^{k} g(e_{\pi_j}) \right) .$$

These are exactly negatives of each other, since they only differ by a power of -1. Thus, any term in Term 1 is cancelled by exactly one term in Term 2. Likewise, each term in Term 2 has a term in Term 1 that cancels it. Thus, there exists a bijection between terms in Term 1 and Term 2 and it follows that the expression evaluates to 0.

Thus, we conclude that (Characters, $*$) is indeed a group. □

A finitely truncated (Characters, $*$) can also be proven to be a **Lie group**.

We now define the **derivation** of $T(\mathbb{R}^d)$ based on the counit $\epsilon \in T(\mathbb{R}^d)^*$. In this case, the derivation Der is

$$\text{Der} = \{f : T(\mathbb{R}^d) \to \mathbb{R} : f \circ ⧢ = \epsilon \otimes f + f \otimes \epsilon\}. \tag{2.30}$$

Der can be proven to be a Lie algebra with Lie bracket defined as the commutator:

$$[f, g]_* = f * g - g * f. \tag{2.31}$$

Because of the Lie algebra-Lie group correspondence, there exists a mapping between these two spaces (the characters group and the derivation). Namely, for any $g \in$ Characters,

$$\log(1 + g) = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{g^{*k}}{k!} \in \text{Der}, \tag{2.32}$$

and in the opposite direction, for all $g \in$ Der,

$$\exp(g) = \sum_{k=0}^{\infty} \frac{g^{*k}}{k!} \in \text{Characters}. \tag{2.33}$$

As a final note linking this algebraic monstrosity back to the signature, the signature mapping is of course a character as it obeys the shuffle product identity. Thus, the log signature is defined as

$$\log(1 + S(X)) = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{S(X)^{*k}}{k!} \tag{2.34}$$

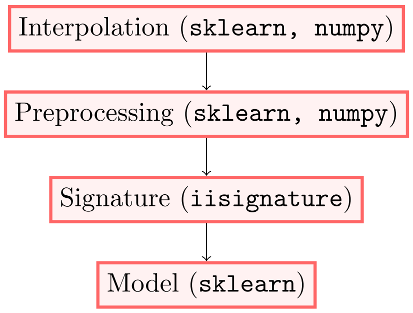
Interpolation (sklearn, numpy)
Preprocessing (sklearn, numpy)
Signature (iisignature)
Model (sklearn)

FIGURE 1. Pipeline overview of model.

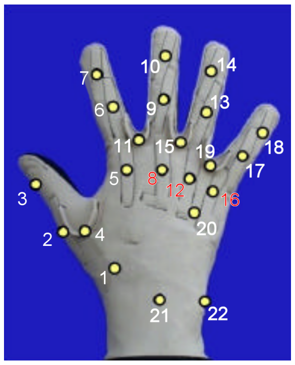

FIGURE 2. Cyberglove sensor mapping (1 indexed).

and lives in the Der Lie algebra.

There are many theoretical advantages of working with the log signature instead of the original signature, but application wise, the log-signature can be thought of as a compact representation of the entire signature (see Table 1), removing the redundancy in the signature exposed by the shuffle product identity. We fit our models onto the log signature instead of the signature itself and justify this decision in subsection 3.3.

3. Experiments

An overview of our model pipeline is displayed in Figure 1.

3.1. **Data.** We use Cyberglove data from DB1 and DB5 taken from the NinaPro datasets 1 and 5 [Jar+19]. The model is the Cyberglove 2, which has 22 sensors located in various positions on the wearer's hand as depicted in Figure 2. In total there are 6 repetitions from 37 intact (non-amputee) subjects. We apply the signature method to attempt to classify different movements of the fingers.

For the small case study done here, we classify between 4 exercises (stimuli $\{1, 3, 5, 7\}$) respectively described as:

(1) index finger flexion,
(3) middle finger flexion,
(5) ring finger flexion,
(7) little finger flexion.

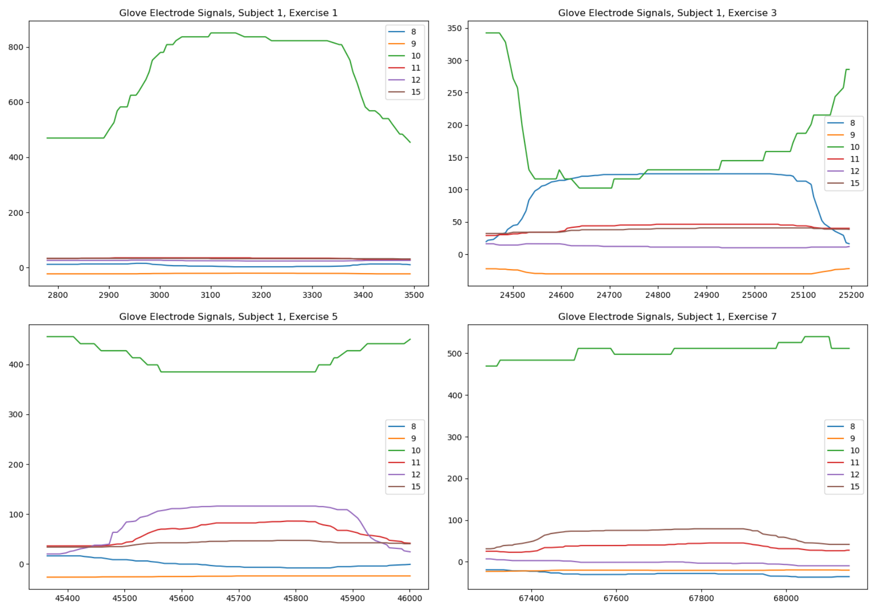

Figure 3. Signals of selected sensors (0-indexed, not 1 indexed like Figure 2) for exercises 1, 3, 5, 7.

We expect to see drastic differences between these movements, namely in which sensors are being activated. Visually, they are completely different movements as shown in Figure 3 and the signature captures this.

3.2. **Preprocessing.** Firstly, the paths are normalized to remove the means and set variances to 1. Normalization is necessary, since as shown in Figure 3, the paths have wildly different means and variances.

3.2.1. *Data Engineering.* Because of a limited set of data (only 37 subjects with 4 training repetitions of each stimulus), we use preprocessing techniques to expand the data. Let $X = \{\mathbf{X}_i\}_{i=1}^{N}$ be the discrete datastream of N points. We provide formal definitions of two methods of engineering additional data here, but emphasize their intuitive nature.

(1) Overlapping windows: following [Jar+19], we construct a set of overlapping windows W parameterized by `window_sz` and `overlap` $\leq \lfloor \texttt{window\_sz}/2 \rfloor$ such that

$$W = \{W_i\}_{i=1}^{M}, \quad M = \left\lceil \frac{N - \texttt{window\_sz}}{\texttt{overlap}} \right\rceil,$$

and W_i is defined as

$$\begin{aligned} W_i &= [X_{r_i}, X_{s_i}], \\ r_i &= (i-1) \cdot (\texttt{window\_sz - overlap}) + 1, \\ s_i &= \min\left(N, r_i + \texttt{window\_sz}\right). \end{aligned}$$

We use `window_sz` $= 200$ and `overlap` $= 100$ for our experiments.

Intuition: slide a "window" of a particular size over the time interval, each

TABLE 1. Dimensions of signature & log-signature up to depth 4.

Dimensions	Signature	Log Signature
22	245410	62238
5	780	205

time moving it by an amount that results in the specified overlap between the new window and the previous window.

(2) Downsampling: we downsample X into M paths $\{W_i\}_{i=1}^{M}$ where M is the parameter. The paths are constructed as

$$W_k = \{X_i : i \bmod M \equiv k \bmod M\}.$$

Clearly these paths will differ by at most 1 in length. In our case, to be comparable to the previous method, we set $M = \lfloor N/200 \rfloor$, so that each path is still around 200 data points long.
Intuition: convert a "high" frequency path into multiple "low" frequency copies.

Remark: Ragged data is not an issue under the signature framework. There are no issues if one of the windows or downsampled paths is not exactly the same size as the others, since the signature acts like a uniform dimensionality reduction with respect to the time axis.

Because of the signature's properties, intuitively (2) is expected to perform better because it preserves the overall shape of the path, whereas (1) potentially results in many similarly shaped paths that are labelled differently. We confirm this suspicion experimentally in subsection 4.2.

3.2.2. *Dataset Split.* We split the dataset into 2 sets: training and testing (which is different than the traditional train-validation-test split because of limited data) using the same method as [Jar+19]: repetitions $\{1, 3, 4, 6\}$ are used as training data while repetitions $\{2, 5\}$ are used for testing. Furthermore, the path data is normalized so that the empirical expectation is 0 and variance is 1.

After applying preprocessing method (1), there are 2204 training paths and 1065 testing paths whereas applying preprocessing method (2) results in 1241 training data points and 609 testing data points.

3.3. **Limitations of Signatures.**

The main limitation of applying signatures is that the number of features grows exponentially with respect to d. For example, suppose we wanted to compute the signature up to depth 4, which is a reasonable depth (1 and 2 are pretty trivial values). We display the values in Table 1.

It is extremely apparent that if we were to use the entire glove data (22 dimensions), trying to apply machine learning would not be computationally feasible nor would it be very meaningful - our model would overfit completely because the complexity of the hypothesis class is so vast. Thus, we introduce bias into the system through a sort of "reverse feature engineering": instead of using the entire glove data, we selectively choose the features that have the greatest variation. For the selected exercises, this intentionally ends up being the features displayed in Figure 3.

To even further reduce the dimensions, we use the log signature instead of the signature itself. We remark that in theory the shuffle product identity no longer holds when using the log signature - however, we determine that the performance gained from reducing dimensionality of the feature space, particularly for generalizing to unseen data, outweighs having the linear relationships.

For more theoretical justification as to why this is a reasonable choice, we note that the number of terms in $X^{(I)} \sqcup X^{(J)}$ is exactly $\binom{|I|+|J|}{|I|}$, which grows quickly as $|I|$ and $|J|$ grow. Thus, the beautiful idea that any polynomial of signature terms can be expressed as a linear combination of higher order terms lives in the vast realm of theoretical results that are "pretty, but computationally intractable."

3.4. **Relevant Models.** Despite the reduction in dimensions through reverse feature engineering, we must use methods that are robust against overfitting in high dimensional feature spaces since the dimension is still large relative to the limited dataset size. We survey 3 methods from simplest to most complex: regularized linear models (ElasticNet), support vector machines, and basic MLPs. For all, we use their respective implementations in `sklearn` and investigate hyperparameter tuning through cross validation.

3.4.1. *ElasticNet.* We recall that ElasticNet is a linear model that attempts to minimize the following combined L^1 and L^2 regularized least squares objective for $w \in \mathbb{R}^d$ (in a binary classification problem):

$$\arg\min_{w} \frac{1}{2n}||y - Xw||_2^2 + \alpha\gamma||w||_1 + \frac{\alpha}{2}(1-\gamma)||w||_2^2, \tag{3.1}$$

where γ is a ratio between the L^1 and L^2 terms and α is the weighting of regularization. Extending to multi class just turns w to a matrix and y becomes a one hot encoded vector of the true class i.e. if label j is the correct label, then the y for that training example looks like

$$y = (\underbrace{0, 0, \ldots, 0}_{j-1 \text{ zeros}}, 1, 0, \ldots 0).$$

Clearly, because we are attempting to minimize the objective in Equation 3.1, we see that the regularization terms *penalize* the weights. Namely, the L^1 regularization ($\alpha\gamma||w||_1$ term) injects sparsity into the weights, as all weights are penalized toward 0, while the L^2 regularization ($(\alpha/2)(1-\gamma)||w||_2^2$ term) ensures smaller weights, ensuring that we don't overuse one feature. Notice, that the regularizations have different effects - mainly, L^2 *does not* force weights toward 0 since if $|w_i| \leq 1$, then $w_i^2 \leq |w_i|$.

Regularization helps prevent overfitting when the dimensionality is high, such as in our case. The disadvantage of using a simple model is that the model complexity is low (it can only fit linear functions of the signature terms), so we could easily underfit the data as well. Luckily, however, we find this empirically is not the case. However, we still survey more complex models as well.

3.4.2. *Support Vector Machine (SVM).* SVMs attempt to fit a model that maximizes the *margin* of the decision boundaries - the margin is the minimum distance between a point in the dataset and the decision boundary. Otherwise, SVMs are very similar to other classification models. For example, Figure 4 compares two model fits with 0 empirical risk on the same dataset, but with vastly different margins.

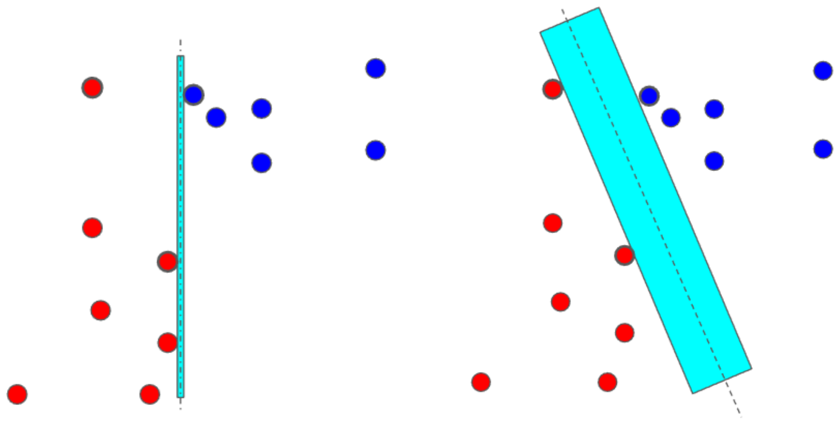

FIGURE 4. Comparison of small vs. large margin on the same dataset using a halfspace decision boundary. Clearly, we prefer the right figure. Images courtesy of DATA2060 taught at Brown University in Spring 2023.

Because of the additional objective to maximize the margin, SVMs are useful even in high dimensional feature spaces, in particular when the number of features is large in comparison to the dataset size. Furthermore, the model complexity is only limited by what function is used for the decision boundary, determined by the **kernel** function of a support vector machine. There has been substantial work deriving a signature kernel [CLX21], but for ease of implementation it is not used here. In fact, it should be considered interesting that an unintegrated approach like ours works so well, speaking to the power of the signature.

We use an SVM objective that is also L^2 regularized with parameter C representing the strength, where as C is inversely proportional to the regularization strength.

3.4.3. *Multilayer Perceptron (MLP).* The final surveyed model is the MLP Classifier, which consists of a feed-forward neural network that maps input data to a set of appropriate outputs. Several connected layers may be used in sequence with a non-linear activation function connecting successive layers. Two additional layers that take in input and return outputs are used as the first and last operations in evaluation of the neural network. Since MLPs are fully connected, each node in one layer connects with a certain weight w_{ij} to every node in the following layer. More concretely, a feed-forward neural network with a single hidden layer of $n \in \mathbb{N}$ neurons defines a function $u^{NN} : \mathbb{R}^d \to \mathbb{R}$ as

$$u^{NN}(x;\theta) = \sum_{j=1}^{n} c_j \sigma(x \cdot W_j + b_j) \tag{3.2}$$

where $d \in \mathbb{N}$ is the fixed dimension of the input data, $W_j \in \mathbb{R}^d, b_j \in \mathbb{R}$ are non-linear parameters, $c_j \in \mathbb{R}$ are linear parameters, and $\sigma : \mathbb{R} \to \mathbb{R}$ is a smooth, bounded function. The set of non-linear and linear parameters is denoted $\theta = \{W, b, c\}$.

Learning consists of altering θ as the neural network is evaluated on each piece of data. Updates are performed based on the amount of error in the output of

TABLE 2. Accuracies for the 3 surveyed models and parameters, signature depth 4.

Model	Train Accuracy	Test Accuracy
ElasticNet ($\alpha = 0.1, \gamma = 0.5$)	0.813	0.757
ElasticNet ($\alpha = 0.05, \gamma = 1.0$)	0.812	0.757
Support Vector Machines ($C = 1.5$)	0.923	0.818
MLP (1 layer, 500)	0.989	0.869

the neural network compared to the expected result. This process involves a loss function and an optimizer, where the former is oftentimes tailored to a specific problem domain and the latter is chosen from a variety of standard algorithms. For our application, we use the standard Adam optimizer [KB17] and utilize log-loss (equivalently logistic or cross-entropy loss), a standard supervised learning classification loss function.

4. Results

The accuracies for the optimal configurations of each model with signature depth 4 determined through cross validation are displayed in Table 2. We see that the MLP, even with just one hidden layer and 500 nodes performs best, providing hope that with more in depth analysis a powerful neural network can be trained to predict at even higher accuracies.

Cross validation results are shown in Figure 5 for ElasticNet, Figure 6 for support vector machines, and Figure 7 for the MLP. The results on unseen data are also displayed in the confusion matrices in Figure 9.

There are a couple explanations for why we believe the neural network performs better than the other models. Firstly, the model complexity is much greater in a neural network, hinting that we could be slightly underfitting in the other models. Secondly, we note that the neural network's intermediate layer "softens" the model's compression down into one predicted label. ElasticNet and SVM have to perform a difficult 1-step dimensionality reduction from 200 dimensions down to 1 predicted label, while the neural network has the luxury of additional layer in the middle to convey more information.

4.1. Cross Validation of Depth of Signature.

We compare all depths of the signature for the SVM model. We notice that despite depth 4 containing the same information as depth 3, the accuracy is worse. Depth 3 peaks at over 0.87 testing accuracy. However, we do not note this same performance jump in the other models - ElasticNet performs worse on training data but the same on the testing data, while MLP performs the same on both datasets.

Overall, these results hint that most of the information necessary to classify the paths is actually present in a subset of the signature. As of writing, intuition on how to select the depth of the signature is not well known or studied - it should be tuned using cross validation.

4.2. Preprocessing Method Comparison.

We also ran the same models with preprocessing method 1. As expected, the performance is significantly worse - a vanilla RBF kernel support vector machine is only able to achieve 0.63 testing accuracy.

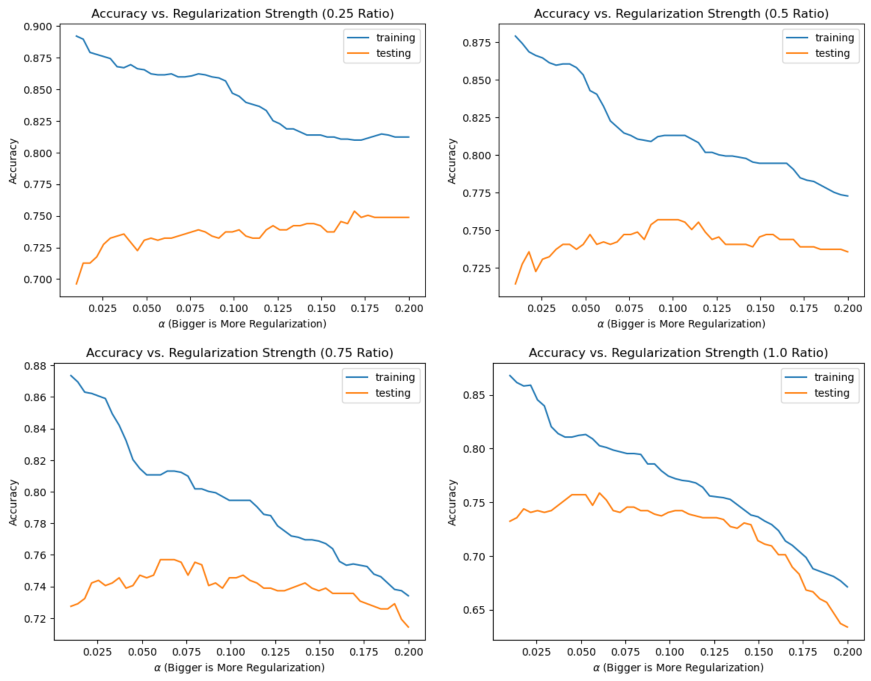

FIGURE 5. Accuracy of ElasticNet with respect to the L^1 ratio and α (regularization strength). When the L^1 ratio is 1.0, we recall that ElasticNet is equivalent to Lasso Regression.

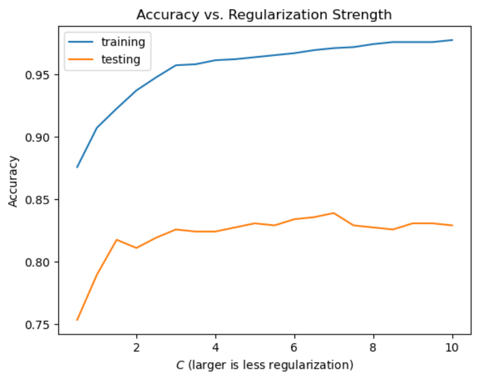

FIGURE 6. Accuracy of C-support vector machine model, RBF kernel, with respect to regularization strength. Signature depth 4.

Looking at the confusion matrices in Figure 10, we can see that the model learns to predict stimuli 3 a lot, instead of the correct label, likely because 3 is the most represented label in the training dataset. Taking a look at paths in Figure 3, we can see that there are likely very similar paths (shapewise) that result from the overlapping windows preprocessing, even though they have different labels, such as in the flatter sections, and the model is unable to distinguish well between them.

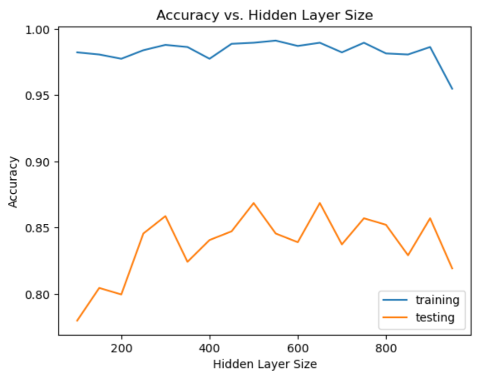

FIGURE 7. Accuracy of 1 layer MLP with respect to the hidden layer size.

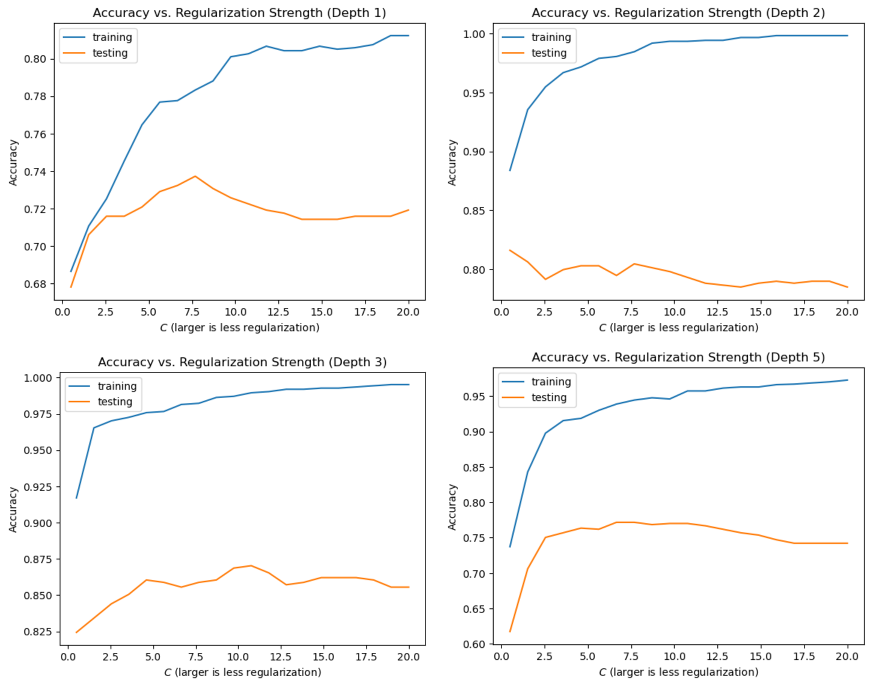

FIGURE 8. Cross validation with SVM rbf kernel with respect to depth of signature and regularization.

4.3. **Sensitivity to Additional Dimensions.** As an additional experiment, we add in two more dimensions of the path to see how the performance of the models are affected. We restrict to just using the SVM as an example.

The two dimensions added were 2 and 20, chosen arbitrarily. The addition of these features increases the dimension of the log-signature feature space to 728. We use two methods to counteract the additional features: increasing strength of regularization, and analysis of variance (ANOVA) feature selection.

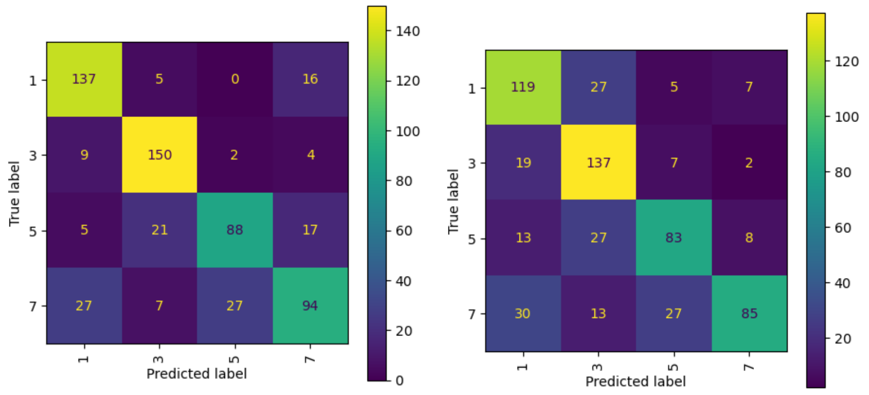

FIGURE 9. Confusion matrices for test data using preprocessing method (2) for SVM (left) and ElasticNet (right).

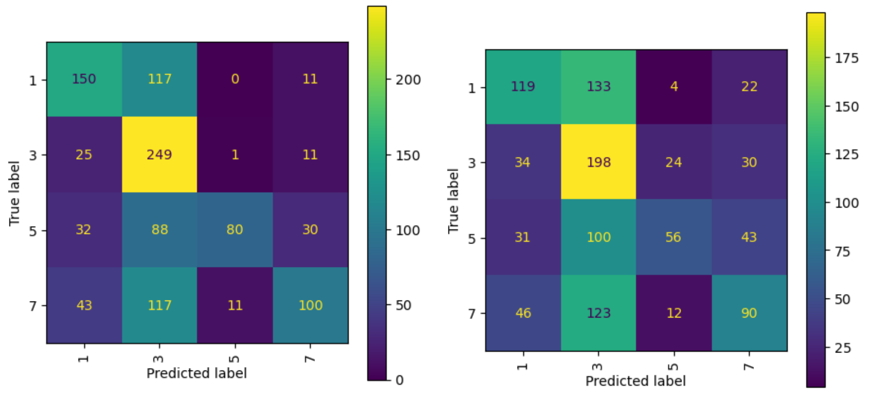

FIGURE 10. Confusion matrices using preprocessing method (1) and SVM (left) and ElasticNet (right).

By decreasing the strength of regularization to $C = 3.0$, the SVM is able to maintain a reasonable accuracies of 0.965 and 0.701 for training and testing respectively, clearly completely overfit. This is likely because the current SVM objective does not have any L^1 regularization that would force sparsity in the weights.

On the other hand, by performing F distribution based feature selection prior to a regularized SVM, we are able to improve this performance. The results are presented in Figure 11 along with standard deviations of - clearly, we can observe that cross validation score when we take 10% of the features easily exceeds the model with stronger regularization.

The end goal of this style of sensitivity analysis is to find a tractable method for avoiding manual feature selection. With the current pipeline (Figure 1), it is not obvious how this can be done and is the greatest limitation of the current work. Most likely, it will involve implementing ideas like the aforementioned signature kernel.

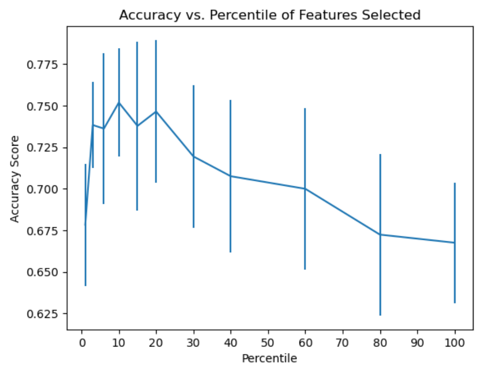

FIGURE 11. Accuracy of SVM + ANOVA against percentage of features selected. Standard deviations are also included at each percentage tested.

4.4. **Comparison to Previous Work.** The current state of the art in the field is using principal component analysis such as in [Jar+19]. Unfortunately, they do not provide accuracies on out of sample data and instead evaluate the performance based on the explained variance of the principal components - this is a strength of our pipeline, as we are able to explicitly display how we expect our model to perform in the wild.

However, a limitation of this work compared to PCA is the lack of interpretability. The principal components that arise from performing PCA are easily interpretable as “synergies” between dimensions. The mathematical interpretations of signature terms are only well understood up to depth 2 as signed areas. However, because signatures are a relatively new method compared to PCA, we can expect this interpretability issue to be resolved in future works.

5. Future Work

Future work includes investigating further the problem of somehow embedding the entire 22 dimensional path. With how quickly the dimensionality increases, robust feature selection is absolutely necessary if the entire truncated log-signature or signature is to be used.

One possible remedy is implementing the signature kernel as aforementioned [CLX21]. Kernel methods allow for infinite dimensional embeddings, meaning that this semi-curse of dimensionality that we encountered would not be an issue.

All subjects analyzed in this paper were intact subjects. However, a natural extension of this work is to amputees. Identifying the movement that an amputee is attempting to perform is the core problem in designing prosthetics. However, a few difficulties exist. Firstly, with amputees, you cannot use Cyberglove data since they don’t have a hand to put the glove on - this means we must use different data, such as sEMG data recorded from the forearm.

This data is visibly noisier and much less interpretable. Filtering can be applied, but it is still hard to extract too much meaning (Figure 12). With how jagged the paths are, it could be interesting to apply the signature theory with *rough paths* instead of assuming nice, smooth paths like in this project.

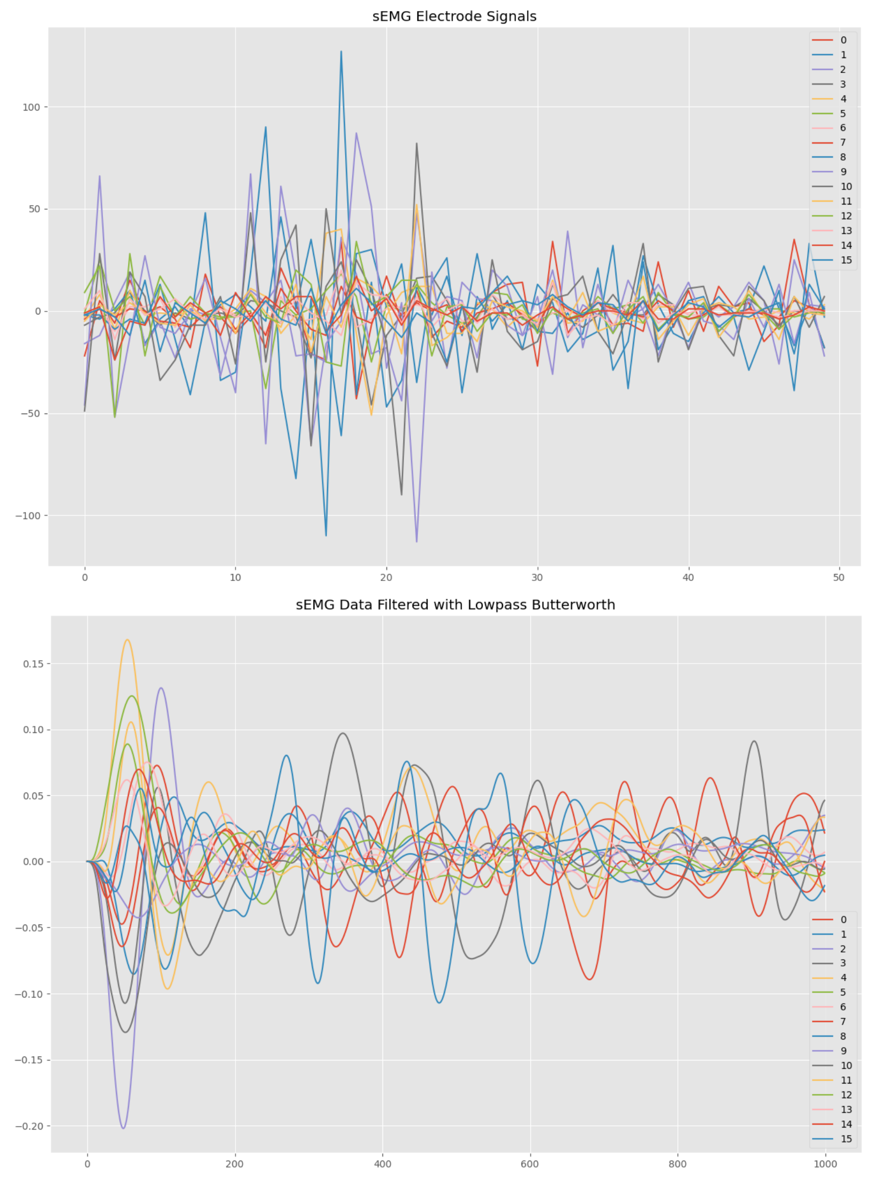

Figure 12. Raw sEMG data compared with normalized sEMG data filtered with a lowpass Butterworth filter.

Further, we note the simplicity of our currently examined models. With larger scale models, the signature has in the past proven to be an extremely effective set of features to use [YJL15] - we primarily confirm through a small case study that it is meaningful to continue investigating the application of signatures to this problem.

6. Appendix

6.1. **Code.** All code used for this project is located in alizma/signatures-hand-synergies.

References

[Ree58] Rimhak Ree. "Lie Elements and an Algebra Associated With Shuffles". In: *Annals of Mathematics* 68.2 (1958), pp. 210–220. ISSN: 0003486X. URL: `http://www.jstor.org/stable/1970243` (visited on 04/17/2023).

[SFS98] Marco Santello, Martha Flanders, and John F. Soechting. "Postural hand synergies for tool use". In: *The Journal of Neuroscience* 18.23 (1998), pp. 10105–10115. DOI: `10.1523/jneurosci.18-23-10105.1998`.

[HK14] Martin Hairer and David Kelly. *Geometric versus non-geometric rough paths*. 2014. arXiv: `1210.6294 [math.PR]`.

[YJL15] Weixin Yang, Lianwen Jin, and Manfei Liu. *DeepWriterID: An End-to-end Online Text-independent Writer Identification System*. 2015. arXiv: `1508.04945 [cs.CV]`.

[LLN16] Daniel Levin, Terry Lyons, and Hao Ni. *Learning from the past, predicting the statistics for the future, learning an evolving system*. 2016. arXiv: `1309.0260 [q-fin.ST]`.

[KB17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[Arr+18] Imanol Perez Arribas et al. "A signature-based machine learning model for distinguishing bipolar disorder and borderline personality disorder". In: *Translational Psychiatry* 8.1 (Dec. 2018). DOI: `10.1038/s41398-018-0334-0`. URL: `https://doi.org/10.1038%2Fs41398-018-0334-0`.

[Jar+19] Néstor J. Jarque-Bou et al. "Kinematic synergies of hand grasps: A comprehensive study on a large publicly available dataset". In: *Journal of NeuroEngineering and Rehabilitation* 16.1 (2019). DOI: `10.1186/s12984-019-0536-6`.

[RG20] Jeremy F. Reizenstein and Benjamin Graham. "Algorithm 1004". In: *ACM Transactions on Mathematical Software* 46.1 (2020), pp. 1–21. DOI: `10.1145/3371237`.

[CLX21] Thomas Cass, Terry Lyons, and Xingcheng Xu. *General Signature Kernels*. 2021. arXiv: `2107.00447 [math.PR]`.