Nullspace-based metric for classification of dynamical systems and sensors

Dominique Martinez, Mohamed Boutayeb

Nullspace-based metric for classification of dynamical systems and sensors

Dominique Martinez[1] and Mohamed Boutayeb[2]

[1] Aix-Marseille Université, CNRS, ISM, 13009 Marseille, France,
dominique.martinez@univ-amu.fr
[2] Lorraine Université, CNRS, INRIA, CRAN, 54506 Nancy, France,
mohamed.boutayeb@univ-lorraine.fr

Abstract. A growing field in machine learning is to develop computationally friendly algorithms optimized for classification of sensor data at the edge. As many sensors respond to different stimuli with different dynamics, the sensor dynamics may provide valuable information for classification. The problem of determining which of the dynamics has generated a particular observation vector is refered to as a multiclass discrimination problem. A discriminative metric can be computed in a Kronecker space provided the different dynamics are represented with state-space models having distinct observability matrices. In this paper, we express the metric in original space rather than in Kronecker space. One of the main contributions is to show theoretically that the metric for a given system is provided by the norm of the matrix-vector product between the left nullspace of the observability matrix and the observation vector. In practice, when the system is unknown and noisy, an "approximate" nullspace is obtained with a data-driven approach using eigenvalue or singular value decomposition. We tested the classifier on synthetic and real datasets. Results demonstrate the applicability of the method. The new formulation of the metric in original space leads to a classifier with reduced complexity so that learning can be implemented with a few lines of code. This low complexity is suitable for machine learning on low-cost, low-power, and resource-constrained devices like smartphones and sensors.

Keywords: Classification · Dynamical systems · machine Learning · State-space models · Nullspace · Observability · Time series · Edge computing · TinyML

1 Introduction

With the multiplication of sensors and IoT devices in our daily life, there is a growing interest for lightweight machine learning (ML) or tiny ML capable of performing sensor data classification at the edge, thus avoiding the need to transfer data over long distances [1]. Deep learning [2] is not the ideal candidate for classification on embedded systems as it requires a large number of training data combined with lots of computational and memory resources to train

and store a huge number of parameters. In contrast, traditional ML is less expensive in terms of memory and computation. This is particularily the case for distance-based classifiers, such as k-nearest neighbors [3], k-means [5] or Parzen windows [4]), which are the simplest classification algorithms. The classification rule during the inference process is based on the dissimilarity, as measured by a distance function, between input samples and prototypes in the training set. Distance-based methods have been developped originally for static data using the Euclidean or Mahalanobis distance. Yet, sensor data are usually not static but evolves in time and the sensor dynamics may provide valuable information for classification. Chemical sensors, for example, react with different dynamics to different odorants and the sensor dynamics can be exploited to discriminate among odorants [6].

Here, one considers that sensor data from a given class are generated by some underlying dynamical system represented with a state-space model. State-space modeling is an effective tool to represent dynamical systems. A variety of artificial and natural systems have been described precisely by state-space models, e.g. the control of an aircraft that depends on the flight conditions [7], the response of gas sensors to different chemical stimuli [8] or the behaviour of animals relative to their environmental conditions [9]. In a state-space model, it is assumed that the state of the system evolves in time but cannot be observed directly. There are two equations. One is the transition equation which predicts the future state given the current state and the other is the measurement equation which provides the observations conditional on the current state. The system is said to be observable in a control-theoric sense if all its states can be retrieved from the sensor data, a condition that is satisfied when the observability matrix has full rank.

Recently, it has been shown that a classifier can be designed to discriminate between different state-space models from sensor data, even when the systems are not observable, provided their observability matrices are all different [10]. We also proposed an appropriate metric to quantify whether sensor data result from a particular state-space model or class. Yet, the distance is computed in a high-dimensional Kronecker space with quadratic complexity which may prevent its use for embedded classification on small devices. In this paper, we revisited the minimum distance classifier for dynamical systems to limit complexity. We show theoretically that the distance can be written in original space as a sum of dot products with a set of weight vectors that corresponds to the left nullspace of the observability matrix. This new formulation leads to a distance-based classifier with reduced learning complexity which makes it suitable for machine learning on embedded devices.

The rest of the paper is organized as follows. The classification problem is specified and described in Section II for dynamical systems. Classifier inference, that is distance computation in original space, is presented in Section III with main result provided by Theorem 2. Classifier learning using eigenvalue or sin-

gular value decomposition in case of unknown dynamical systems is presented Section IV. To demonstrate its simplicity, we implemented the learning process as one line of code in matlab (see Note 3). Results obtained on synthetic and real datasets are presented in Section V. Finally, the main conclusions are presented in Section VI.

2 Problem statement

Throughout this paper, the following notation is used. All vectors are column vectors unless transposed to a row vector by a prime notation $'$. We consider a dynamical system represented as the following state-space model:

$$\begin{aligned} x(k+1) &= Ax(k) \\ y(k) &= Cx(k) \end{aligned} \tag{1}$$

with state $x \in \mathbb{R}^{n\times 1}$, output $y \in \mathbb{R}^{m\times 1}$, transition matrix $A \in \mathbb{R}^{n\times n}$ and output matrix $C \in \mathbb{R}^{m\times n}$.

Let us further consider a vector of N consecutive observations written as $Y \in \mathbb{R}^{l\times 1}$ with $l = Nm$:

$$Y = \begin{pmatrix} y(k) \\ y(k+1) \\ \cdots \\ y(k+N-1) \end{pmatrix} \tag{2}$$

The problem is to find an appropriate distance $d(Y)$ between Y and system (1) so that it can be applied for the classification of dynamical systems. For example, in one-class classification, (1) would describe the normal operating mode of the system. Unusual events (e.g. novelty, anomaly or fault events) would be detected whenever $d(Y)$ exceeds a predetermined threshold. In multi-class classification, one has different dynamical models (1) and the classifier would assign the observation vector Y to the one with the smallest distance.

3 Inference (distance computation)

If Y was generated by system (1) then we have:

$$Y = O_b\, x(k) \tag{3}$$

with O_b the observability matrix defined as:

$$O_b = \begin{pmatrix} C \\ CA \\ \cdots \\ CA^{N-1} \end{pmatrix} \tag{4}$$

Even if system (1) is not observable in a control-theoric sense (i.e. O_b is not full rank), $x(k)$ can be determined from (3) by using the pseudo-inverse of O_b, denoted O_b^+. The entire set of solutions is represented by:

$$x(k) = O_b^+ Y + (I - O_b^+ O_b)\Gamma \tag{5}$$

with the arbitary vector Γ varying over all possible values [14].

The distance between any observed sequence Y and system (1) writes as follows [10]:

$$\begin{aligned} d^2(Y) &= ||Y - O_b x(k)||^2 \\ &= ||Y - O_b O_b^+ Y - (O_b - O_b O_b^+ O_b)\Gamma||^2 \text{ (from 5)} \\ &= ||Y - O_b O_b^+ Y||^2 \text{ (as } O_b O_b^+ O_b = O_b) \\ &= [(I-Q)Y]'[(I-Q)Y] \text{ (with } Q \triangleq O_b O_b^+) \\ &= Y'(I - Q' - Q + Q'Q)Y \\ &= Y'(I-Q)Y \text{ (as } Q = Q' \text{ and } Q'Q = Q) \end{aligned}$$

Thus the distance between any observed sequence Y and system (1) writes

$$d^2(Y) = Y'MY \tag{6}$$

With $M = (I - O_b O_b^+) \in \mathbb{R}^{l \times l}$. The matrix equation (6) is reshaped into a vector equation by using the vectorization operator $vec()$ and Kronecker product $\otimes$.

Theorem 1. Distance in Kronecker space.

$$\begin{aligned} &d^2(Y) = Y'MY \Leftrightarrow d^2(Y) = w'(Y \otimes Y) \\ &\textit{with } w = vec(M) \in \mathbb{R}^{l^2 \times 1} \end{aligned}$$

Proof.

$$\begin{aligned} d^2(Y) &= Y'MY \\ &= vec(Y'MY) \\ &= (Y' \otimes Y')vec(M) \\ &= vec(M)'(Y \otimes Y) \\ &= w'(Y \otimes Y) \end{aligned} \tag{7}$$

Note 1. In Matlab, the vec operation can be computed as **w=M(:)** and the Kronecker product is implemented as **kron(Y,Y)**.

Although (7) is a simple scalar product, the Kronecker product requires the computation of l^2 products between all pairs of the l components of Y. Yet, Theorem 2 allows us to limit complexity by computing the distance in original space rather than in Kronecker space.

Theorem 2. Distance in original space.

$$d^2(Y) = Y'MY \Leftrightarrow d(Y) = ||V'Y||$$
$$with\ V = Null(O'_b) \in \mathbb{R}^{l \times k}$$

where Null denotes the nullspace and $k = l - rank(O_b)$.

Proof. To avoid computing the distance in Kronecker space, w is rewriten as a sum of Kronecker products:

$$w = vec(M) = \sum_{i=1}^{k} v_i \otimes v_i$$

with weight vectors $v_i \in \mathbb{R}^{l \times 1}, i = 1 \cdots k$.
The distance then becomes as follow

$$\begin{aligned} d^2(Y) &= w'(Y \otimes Y) \\ &= \sum_{i=1}^{k} (v'_i \otimes v'_i)(Y \otimes Y) \\ &= \sum_{i=1}^{k} (v'_i Y) \otimes (v'_i Y) \\ &= \sum_{i=1}^{k} (v'_i Y)^2 \\ &= ||V'Y||^2 \text{ with } V = (v_1 \cdots v_k) \end{aligned} \tag{8}$$

For Y generated by system (1), $d^2(Y) = 0$ leads to

$$\begin{aligned} \sum_{i=1}^{k} (v'_i Y)^2 &= 0 \\ \Leftrightarrow v'_i Y &= 0\ \forall i \\ \Leftrightarrow Y' v_i &= 0\ \forall i \\ \Leftrightarrow x'(k) O'_b v_i &= 0\ \forall i\ \forall x(k) \neq 0 \\ \Leftrightarrow O'_b v_i &= 0\ \forall i \\ \Leftrightarrow v_i &\in Null(O'_b) \\ &\text{for } i = 1 \cdots k = l - rank(O_b) \end{aligned}$$

Note 2. The distance is now expressed as the norm of a matrix-vector product in the original space Y, as stated by (8), and not in the Kronecker space as in (7). In Matlab, it can be implemented simply as **d=norm(**$V' * Y$**)** with **V=Null(**O'_b**)**

4 Learning (approximate nullspace)

When computing the distance in original space, the weight vectors v_i's can be obtained from the left nullspace of the observability matrix (Theorem 2). A prerequisite, however, is that the dynamical system is known exactly. In case it is unknown or partially known, the v_i's have to be estimated from training data. Let $Y_{train} = (Y^1, \cdots, Y^p) \in \mathbb{R}^{l \times p}$ be the training set corresponding to the concatenation of p observation vectors Y^μ, $\mu = 1 \cdots p$, for system (1). Ideally, $d(Y^\mu)$ should be zero for all μ so that $Y'_{train} v_i$ is a null vector for all i. Thus, v_i belongs to the nullspace of Y'_{train}. In practice, however, the observations are corrupted by noise so that the nullspace of Y'_{train} does not necessarily exist. We therefore look for an "approximate" nullspace by considering the span of the left singular vectors associated with the k smallest singular values, with $k = l - rank(O_b)$. The solution can be obtained through singular value decomposition (SVD) or eigenvalue decomposition as the left singular vectors of Y_{train} correspond to the eigenvectors of the sample autocorrelation matrix $Y_{train} Y'_{train} = \sum_{\mu=1}^{p} Y^\mu (Y^\mu)'$. There are a number of algorithms that can be apply to extract the k minor components of the sample autocorrelation matrix [11], including online algorithms to update the v_i's as new observations become available [12].

Note 3. In Matlab, learning can be implemented as one line of code : **[V,** $\sim$**] = eigs(**Y_{train}*****Y'_{train}**, k, 'smallestabs')** or **[V,** $\sim$**,** $\sim$**] = svds(**Y_{train}**, k, 'smallest')**.

Alternatively, one might consider all data for training and not only those of the class under consideration. Let $\overline{Y}_{train}$ the training data of the other classes. Ideally, $d(Y)$ should be small for Y_{train} and large for $\overline{Y}_{train}$. This strategy leads to minimizing the Rayleigh quotient defined as $||V' Y_{train}||^2 / ||V' \overline{Y}_{train}||^2$. The solution is obtained by solving a generalized eigenvalue problem with matrices $(Y_{train} Y'_{train})$ and $(\overline{Y}_{train} \overline{Y}'_{train})$ in which V is the set of the k generalized eigenvectors that corresponds to the smallest eigenvalues.

5 Experiments

The nullspace-based classifier is tested on synthetic and real datasets. We compare the data-driven approach using SVD learning, as in Note 3, to the model based approach using system identification or a priori knowledge. When the exact knowledge of the system is available, the ground truth is the best possible classifier using the left nullspace of the observability matrix as in Note 2.

5.1 XOR problem of dynamical systems

In this illustrative example, one considers the dynamical version of the XOR classification problem. Let $x(k+1) = x(k)$ (class 1) and $x(k+1) = -x(k)$ (class 2) be the two dynamical systems depicted in Fig. 1, with $y(k) = x(k)$ for the two classes. We consider observation vectors with $N = 2$, i.e. $Y = (y(k)\ y(k+1))'$, so that the observability matrices are $O_1 = (1\ 1)'$ (class 1) and $O_2 = (1\ -1)'$ (class 2). We further consider $p = 2$ training samples for each class such that:

$$Y_{train,1} = \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix} \qquad Y_{train,2} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

As seen in Fig. 1, the classification problem is not linearly separable, i.e. $Y_{train,1}$ and $Y_{train,2}$ cannot be separated with a straight line. The nullspaces of O_1' and O_2' resume to single vectors V_1 and V_2. Without observation noise, the data-driven approach leads to the same solution as the ground truth, that is $V_1 = (-0.7071\ 0.7071)'$ and $V_2 = (0.7071\ 0.7071)'$, which correspond to the crosslines in Fig. 1 (left). In the presence of noise, the solution obtained with $p = 10$ training samples is depicted in Fig. 1 (right).

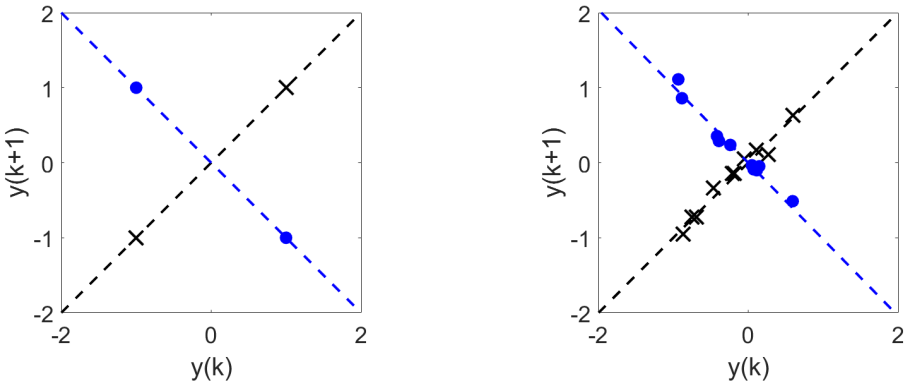

Fig. 1. Nullspace-based classification on the XOR example without noise (left) and with noise (right). The solution obtained with SVD (data-driven approach) is represented by the crosslines. The training data are depicted as crosses for class 1 and circles for class 2.

5.2 Aircraft simulation

The aircraft simulation is derived from [7] and has been used in [10, **?**]. The dynamical equation for class $i = 1 \cdots 4$ which predicts the future state $x_i(k+1)$ given the current state $x_i(k)$ and input $u_i(k)$ is given by

$$x_i(k+1) = \mathcal{A}_i x_i(k) + \mathcal{B}_i u_i(k)$$

and

$$\mathcal{A}_i = \begin{pmatrix} a_{i,11} & a_{i,12} & a_{i,13} \\ a_{i,21} & a_{i,22} & a_{i,23} \\ 0 & 0 & a_{i,33} \end{pmatrix} \qquad \mathcal{B}_i = (b_{i,1}\ b_{i,2}\ b_{i,3})'$$

The entries of matrices $\mathcal{A}_i$ and $\mathcal{B}_i$ depend on the flight condition (speed and altitude). They are given in table 1. They have been obtained by mapping the continuous-time models in [7] to discrete-time models using zero-order hold discretization and sampling time $dt = 0.1$ s. The state x_i has three component: normal acceleration, pitch rate and pitch angle. The input signal u_i is the input to the elevon servo. As the system is unstable for subsonic speeds, a linear quadratic controller is designed using Matlab function *dlqr.m* to improve stability. The feedback controller is written as $u_i(k) = K_i x_i(k)$ with $K_i = (k_{i,1}\ k_{i,2}\ k_{i,3})$ given in Table 1. Thus, the system rewrites as in Eq. 1, i.e. $x_i(k+1) = A_i\ x_i(k)$ with $A_i = \mathcal{A}_i + \mathcal{B}_i K_i$. The problem is to build a classifier that discriminates the different classes ($i = 1 \cdots 4$) from noisy observations $y_i(k) = C_i x_i(k) + \mu$ (μ is a zero-mean gaussian noise).

Class	$i=1$	$i=2$	$i=3$	$i=4$
Mach	0.5	0.85	0.9	1.5
Altitude	5000	5000	35000	35000
a_{11}	0.9268	0.8915	0.9424	0.8648
a_{12}	1.6002	4.4207	1.6990	2.3959
a_{13}	4.3075	8.28	3.7561	6.4622
a_{21}	0.0243	0.0192	0.0077	-0.0613
a_{22}	0.9396	0.9162	0.9432	0.8018
a_{23}	-0.5090	-1.4456	-0.5389	-1.8297
a_{33}	0.2466	0.2466	0.2466	0.2466
b_1	-5.4001	-16.4777	-4.7666	-9.9568
b_2	-0.5931	-1.5991	-0.5101	-0.9536
b_3	0.7534	0.7534	0.7534	0.7534
k_1	0.1556	0.0524	0.1752	0.0788
k_2	0.3710	0.2767	0.4547	0.2664
k_3	0.6478	0.4530	0.6109	0.5204

Table 1. Parameters of the airplane state model for different flight modes (speed and altitude conditions).

We trained the nullspace-based classifier using SVD on $p = 500$ training data for each system, where each training data consists in $N = 4$ consecutive

observations. The error rate is estimated over 1000 test data generated the same way as the training data. As seen in Figure 2, the data driven approach leads to performance similar to the ground truth using the entire nullspace of the observation matrix and to support vector machines (SVMs) [15]. Moreover, the data driven approach outperforms a classifier that uses a single vector of the nullspace, and a model-based classifier built from system identification.

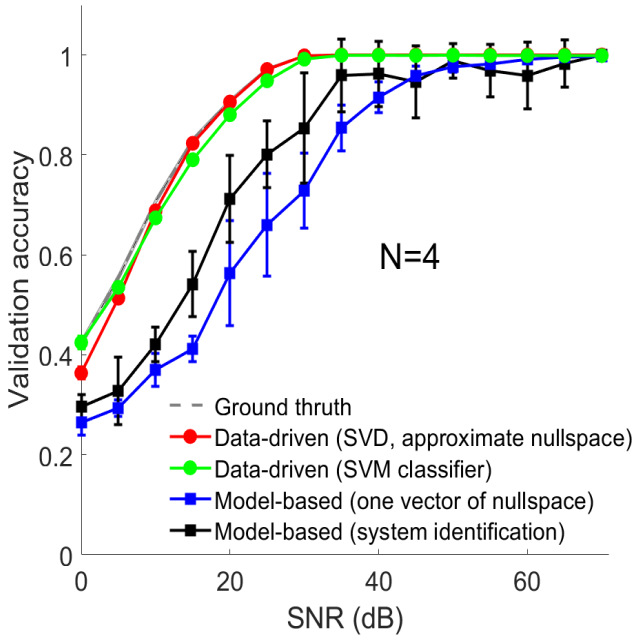

Fig. 2. Classification on the aircraft example. The observation vector consists in $N = 4$ consecutive sensor data. Noise is added to the sensor data. Classifier accuracy is represented versus signal-to-noise ratio (SNR) as mean±S.D estimated over 100 trials. The nullspace-based classifier is built from training data using SVD (note 3). Performance is compared to an SVM classifier and to model-based classifiers (Theorem 2) built from system identification or from the exact knowledge of the system (using the entire nullspace or a single vector). The ground truth (best possible classifier) uses the entire nullspace derived from the exact knowledge of the system.

In theory, the nullspace classifier can be built when the systems are not observable, provided their observability matrices are all different [10]. This condition is statisfied for $N \geq 2$ as the pair (A, C) in (1) is different for different systems. To check the validity of this theoretical condition, we trained the nullspace-based classifier with training data that consist of only $N = 2$ consecutive observations, i.e. $Y = (y(k), y(k+1))'$. As seen in Figure 3, for $N = 2$,

the nullspace-based classifier built using SVD has similar performance to the ground truth and outperforms all other classifiers. It is also worth noting that the system identification approach fails as $N = 2$ is not enough to identify the systems whose state dimension is 3.

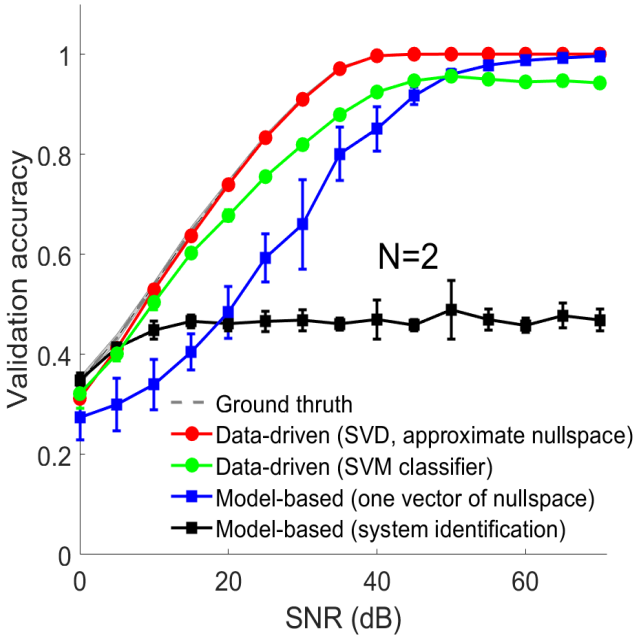

Fig. 3. Nullspace-based classification on the Aircraft example. Same conventions as in Fig. 2 except that the observation vector consists in $N = 2$ consecutive sensor data.

5.3 Handwriting character recognition from pen trajectory

We consider a real dataset from the UCI Machine learning archive[3]. The dataset consists of 2858 handwriting characters (20 classes 'a,b,c,d,e,g,h,l,m,n,o,p,q,r,s,u,v,w,y,z') collected using a WACOM pen tablet. The sensor data consists in pen trajectories with three components recorded at 200 Hz: pen velocity in x-position (mm/s), pen velocity in y-position (mm/s) and rate of the pen tip force (N/s). Figure 4 shows examples of characters reconstructed using the first N=10, 50 and 100 pen trajectory data. We trained the nullspace-based classifier using observation vectors with N=10, 50 and 100. The training set consisted of 10% of the recordings and the validation set consisted of the remaining 90%. Training

[3] `https://archive.ics.uci.edu/dataset/175/character+trajectories`

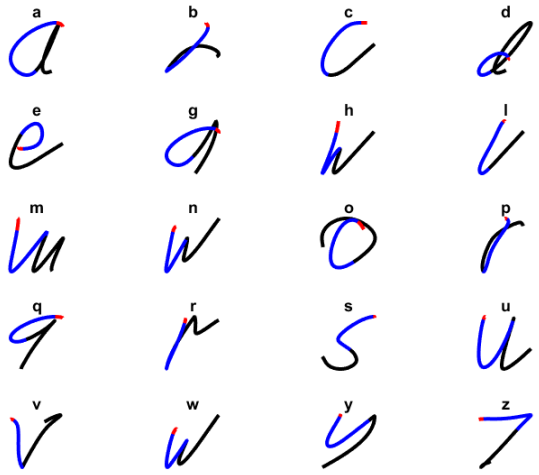

Fig. 4. Reproduction of characters with the first N=10 (plots in red), N=50 (plots in blue) and N=100 (plots in black) sensor data sampled during the pen trajectory. The data are the pen velocity in x-position (mm/s), pen velocity in y-position (mm/s) and rate of the pen tip force (N/s) recorded at 200 Hz.

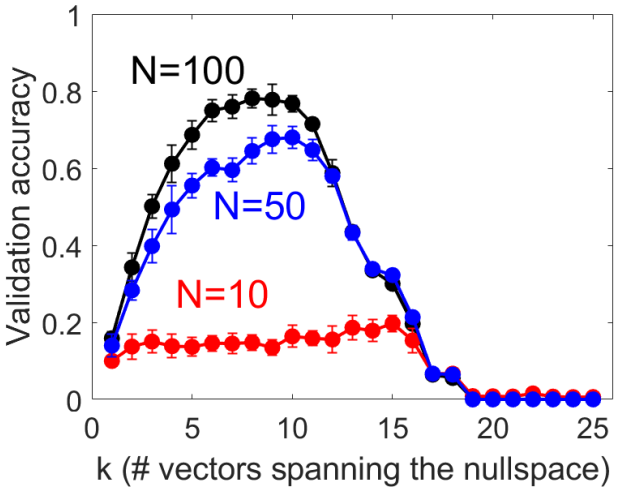

Fig. 5. Nullspace-based classification accuracy with N=100 (plot in dark), N=50 (plot in blue) or N=10 (plot in red) sampled points in the pen trajectory. k is the number of vectors considered for (that span) the nullspace. Results are represented as means±S.D estimated over 10 trials.

and validation were repeated ten times. The validation accuracy, plotted in Figure 5 for different k (number of vectors considered for spanning the nullspace), indicates that optimal performance can be achieved by carefully choosing the dimension of the nullspace.

6 Conclusion

In this paper we propose a nullspace-based classifier for dynamical systems. Computing the metric in Kronecker space vs original space has similar complexity: $\mathcal{O}(l^2)$ vs $\mathcal{O}(kl)$. Indeed, it involves computing a single scalar product of dimension l^2 in Kronecker space vs k scalar products of dimension l in original space. In contrast, learning is more difficult to perform in Kronecker space than in original space as it involves singular or eigenvalue decomposition of a $(l^2 \times l^2)$ sample autocorrelation matrix in Kronecker space vs $(l \times l)$ in original space. For example, in the handwritten character recognition with $N = 100$, one has $l^2 =$ 90,000 in Kronecker space vs $l = 300$ in original space. Still, for a large, near-singular, $(l \times l)$ sample autocorrelation matrix, the "approximate" nullspace is not well defined and can be computationally difficult to extract as it corresponds to eigenvectors associated with very small, but not necessarily zero, eigenvalues. Future work will concentrate in reducing computational requirements on large-scale problems to make it suitable for embedded learning at the edge.

Acknowledgments.This work was funded by France 2030 under the investment program ANR-20-PCPA-0007.

References

1. P.P. Ray, A review on TinyML: State-of-the-art and prospects. Journal of King Saud University - Computer and Information Sciences, 34(4): 1595-1623, 2022
2. Y. LeCun, Y. Bengio and G. Hinton. Deep Learning. Nature, 521:436–444, 2015
3. T. Cover and P. Hart. Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1):21–27, 1967
4. E. Parzen, On estimation of a probability density function and mode. *Ann. Math. Stat*, 33, pp. 1065-1076, 1962.
5. Least squares quantization in PCM, *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129-137, doi: 10.1109/TIT.1982.1056489, 1982.
6. A. Bermak, S.B. Belhouari, M. Shi, D. Martinez, Pattern recognition techniques for odor discrimination in gas sensor array, *Encyclopedia of Sensors*, 2006.
7. K.J. Åström, B. Wittenmark, *Adaptive Control* Reading, MA, USA, Addison-Wesley, 1989.
8. D. Martinez, J. Burgués, S. Marco, Fast measurements with MOX sensors: a least-squares approach to blind deconvolution. *Sensors*, vol. 19, no. 18, doi:10.3390/s19184029, 2019.
9. T.A. Patterson, L. Thomas, C. Wilcox, O. Ovaskainen and J. Matthiopoulos, State–space models of individual animal movement. *Trends in Ecology and Evolution*, vol. 23, no. 2, pp. 87-94, 2008.

10. D. Martinez and M. Boutayeb, Minimum-distance classification for dynamical systems and sensors. *IEEE Sensors Journal*, vol. 23, no. 22, pp. 28454 - 28461, 2023.
11. G.H. Golub, C.F. Van Loan, *Matrix Computations* Third Edition, Johns Hopkins University Press, Baltimore, Maryland, USA, 1996.
12. A. Cichocki, S-I. Amari, *Adaptive blind signal and image processing* John Wiley & Sons, Chichester, England, 2002.
13. G. Battistelli and P. Tesi, Classification for Dynamical Systems: Model-Based and Data-Driven Approaches. *IEEE Transactions on Automatic Control*, vol. 6, pp. 1741-1748, 2021.
14. M. James, The generalised inverse. *Mathematical Gazette*, 62, 420, doi:10.1017/S0025557200086460, 1978.
15. C.J. Burges, A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery* 2, 121–167, doi:10.1023/A:1009715923555, 1998.