Mini-Project (ML for Time Series) - MVA 2023/2024

Time warp invariant kSVD:
Sparse coding and dictionary learning for time series under time warp

Manal Akhannouss manal.akhannouss@eleves.enpc.fr
Alexandre Lutt alexandre.lutt@eleves.enpc.fr

January 9, 2024

1 Introduction and contributions

In this project, we aim to provide an open-source implementation of the Time Wrap Invariant Orthogonal Matching Pursuit and Time Wrap Invariant-kSVD algorithms introduced in [5], and to reproduce some of the experimental results of the original paper.

The classical sparse coding and dictionary learning such as Orthogonal Matching Pursuit (OMP) and kSVD have been widely used in the field of time series analysis. However, these methods are not robust to time warp, which is a common phenomenon in time series. This led the authors of [5] to propose new sparse coding and dictionary learning algorithms that are robust to time warp and allow using variable-length time series. The original paper introduces two algorithms, Time Warp Invariant Orthogonal Matching Pursuit (TWI-OMP) and Time Warp Invariant kSVD (TWI-kSVD). The main idea behind these algorithms is to use a dynamic programming approach to compute the optimal alignment between the atoms of the dictionary and the samples.

During this project, we decided to split the work in two parts. First, Alexandre implemented the different algorithms, then Manal ran experiments to try to reproduce the claimed results. It is worth noting that no codebase was provided by the authors, and that we had to implement everything from scratch. Our joined experiments and tests also led us to propose a new classification strategy to challenge the original one. We also had to make some assumptions regarding the implementation of the algorithms, as well as the preprocessing of the datasets, as many methodological steps were omitted in the paper. Mainly, we had to make assumptions regarding the preprocessing, the size of atoms in dictionaries, and dictionaries initialization. We will discuss these assumptions in more details in the following sections.

2 Method

2.1 Sparse coding

Sparse coding is a method that allows to represent a signal as a linear combination of a few elements of a dictionary. This problem is usually solved with a matching pursuit [1] algorithm such as Orthogonal Matching Pursuit [3] (OMP), which is a greedy algorithm that allows to find the sparse code of a signal by iteratively selecting the atom of the dictionary that is the most correlated with the residual.

Time Warp Invariant Orthogonal Matching Pursuit is a variant of the OMP algorithm [3] that allows to find the sparse code of a signal by iteratively selecting the atom of the dictionary that is the most correlated with the residual, after having computed the optimal alignment between the atoms and the residual. More precisely, the algorithm aims to solve the following:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\Delta}} \quad & \left\| \boldsymbol{x} - \sum_{j=1}^{K} \alpha_j \boldsymbol{\Delta}_j \boldsymbol{d}_j \right\|_2^2 \qquad \text{(TWI Sparse Coding Problem)} \\ \text{s.t.} \quad & \|\boldsymbol{\alpha}\|_0 \leqslant \tau \\ & \boldsymbol{\Delta}_j \mathbf{1}_{p_j} = \mathbf{1}_q, \boldsymbol{\Delta}_j \in \{0,1\}^{q \times p_j} \quad \forall j \in \{1, ..., K\} \end{aligned}$$

where $\boldsymbol{\Delta}_j$ is the optimal alignment matrix between the j-th atom of the dictionary and $\boldsymbol{x}$, $\boldsymbol{d}_j$ is the j-th atom of the dictionary, p_j is the length of the j-th atom of the dictionary, q is the length of $\boldsymbol{x}$, and $\mathbf{1}_n$ is a vector of size n filled with ones. In what follows, we will often use the notation $\boldsymbol{d}_j^s = \boldsymbol{\Delta}_j \boldsymbol{d}_j$ to denote the modified j-th atom of the dictionary that is aligned with $\boldsymbol{x}$.

In order to obtain the optimal alignment matrix $\boldsymbol{\Delta}_j$, the authors of [5] proposed to use a dynamic programming approach to maximise the absolute cosine similarity between the j-th atom of the dictionary and the sample. The optimal alignment matrix between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is found by finding the minimal cost path in a cost matrix $\boldsymbol{C}$, where each element $c_{i,j}$ corresponds to the cosine similarity between the subseries aligned to maximise the cosine similarity. More precisely, $c_{i,j} = \dfrac{\langle \boldsymbol{x}_{i,j} \boldsymbol{y}_{i,j} \rangle}{\|\boldsymbol{x}_{i,j}\|_2 \|\boldsymbol{y}_{i,j}\|_2} = f(M_{i,j})$, where $M_{i,j} = (\langle \boldsymbol{x}_{i,j} \boldsymbol{y}_{i,j} \rangle, \|\boldsymbol{x}_{i,j}\|_2^2, \|\boldsymbol{y}_{i,j}\|_2^2)$ is computed recursively:

$$M_{i,j} = \begin{cases} (\langle \boldsymbol{x}_1 \boldsymbol{y}_1 \rangle, \|\boldsymbol{x}_1\|_2^2, \|\boldsymbol{y}_1\|_2^2) & \text{if } i = 1 \text{ and } j = 1 \\ (\langle \boldsymbol{x}_{i-1,1} \boldsymbol{y}_{i-1,1} \rangle + \boldsymbol{x}_i \boldsymbol{y}_1, \|\boldsymbol{x}_{i-1,1}\|_2^2 + \boldsymbol{x}_i^2, \|\boldsymbol{y}_{i-1,1}\|_2^2 + \boldsymbol{y}_1^2) & \text{if } i > 1 \text{ and } j = 1 \\ (\langle \boldsymbol{x}_{1,j-1} \boldsymbol{y}_{1,j-1} \rangle + \boldsymbol{x}_1 \boldsymbol{y}_j, \|\boldsymbol{x}_{1,j-1}\|_2^2 + \boldsymbol{x}_1^2, \|\boldsymbol{y}_{1,j-1}\|_2^2 + \boldsymbol{y}_j^2) & \text{if } i = 1 \text{ and } j > 1 \\ f(\arg\max(M_{i-1,j}, M_{i,j-1}, M_{i-1,j-1})) & \text{if } i > 1 \text{ and } j > 1 \end{cases}$$

This method is very similar to the DTW algorithm, in the sense that it finds the optimal monotonically increasing alignment that matches beginnings and ends of series together, and computes the optimal alignment matrix between two signals of lengths T_x and T_y in $\mathcal{O}(T_x T_y)$ time. For more details, we refer the reader to Algorithm 1 and [5].

2.2 Dictionary Learning

Dictionary learning is a method that allows to learn a dictionary from a set of signals $\boldsymbol{X}$, as well as the optimal assignments for each signal. The goal of dictionary learning is to find the best dictionary that can represent all signals. We can solve this problem using the kSVD algorithm [2], which is an iterative procedure that aims to learn a dictionary of fixed size by alterning between dictionary updates and sparse coding using the SVD decomposition of the residuals. The global time complexity per iteration of this algorithm is in $\mathcal{O}(TNK\tau)$, where T is the length of a time series.

Time Warp Invariant kSVD is similar to kSVD, but it uses the TWI-OMP algorithm instead of the OMP algorithm to compute the sparse codes. Moreover, the update rule on the atoms and assignments is also changed.

The main idea is to compute the residual errors $\boldsymbol{e}_i$ for each sample $\boldsymbol{x}_i$ that uses the k-th atom $\boldsymbol{d}_k$. We then update $\boldsymbol{d}_k$ using SVD on the rotated residuals $\boldsymbol{\varphi}(\boldsymbol{e}_i)$. The rotation is done using the rotation operator rotation$(\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{c})$, which rotates $\boldsymbol{a}$ so that $\theta_{\hat{a},c} = \theta_{a,b}$. In this case, $\boldsymbol{\varphi}(\boldsymbol{e}_i) =$ rotation$(\boldsymbol{\Delta}_{i,k}^T \boldsymbol{e}_i, \boldsymbol{\Delta}_{i,k}^T \boldsymbol{d}_k^{s_i}, \boldsymbol{d}_k)$. Finally, we apply the reverse transformation to update $\boldsymbol{d}_k^{s_i}$ and $\boldsymbol{\alpha}_{i,k}$. The time complexity per iteration of this algorithm is in $\mathcal{O}(T^2NK\tau)$, where T is the maximum length. More details are provided in Algorithm 2 and [4].

2.3 Evaluation metrics

Original classification method In the original paper, the authors proposed a classification method based on the reconstruction error. More precisely, they proposed to classify a sample $\boldsymbol{x}$ by encoding it using the concatenation of all the class dictionaries, and then by assigning it to the class that gives the lowest reconstruction error. This works under the assumption that only the adequate class atoms will be used to reconstruct the sample with a low reconstruction error. However, if the class atoms are close to each other, this method might not be able to classify the sample correctly.

Proposed classification method During our experiments, we found that we could easily outperform the original classification method by encoding the sample $\boldsymbol{x}$ with each class dictionary separately, and then by assigning it to the class that gives the lowest reconstruction error. We will compare both approaches in the following sections.

3 Data

3.1 Preprocessing

Regarding our data, we used the only two datasets of the original paper that were available online, BME (artificial univariate dataset) and DIGITS (MNIST-like dataset of handwritten digits). As mentioned in the introduction, many details were omitted in the original paper, which led us to make some assumptions regarding the preprocessing of the datasets.

The first step of preprocessing has been to filter out series of the DIGITS dataset that were too long (more than 120 points) or too short (less than 80 points). The main reason behind it was to keep a reasonable computational time, as the TWI-OMP algorithm is quite slow. For the same reason, we preprocessed the BME and DIGITS datasets by respectively downsampling the series to 64 (instead of 128) and 40 to 60 points. Again, in order to keep runtimes reasonable, we decided to respectively keep 10 and 20 series of each class of each dataset for the train and test sets. The last step of our preprocessing has been to standardize the series to the $[-1, 1]$ interval. This was important to ensure that the series were comparable, and that the learned dictionaries would not be biased by the scale of the series.

3.2 Datasets

BME This synthetic dataset consists in univariate time series of fixed size, with 3 balanced classes. The B (Begin) class contains sample with a positive jump at the beginning, the M (Middle) class contains samples without jumps, and the E (End) class contains samples with a positive jump at the end. Samples of each class are shown in Figure 1.

DIGITS This real-world dataset is composed of 2D time series of variable size with 10 balanced classes corresponding to digits handwritten on a digital screen. Samples are shown in Figure 2.

3.3 Experimental setup

Before running the experiments, we had to make some other assumptions regarding the initialization of the dictionaries. Since the original paper only mentioned their lengths (10), we still had to figure out the size of the different atoms, as well as the way to initialize them. After a lot of trial and error, we decided to initialize the atoms of the dictionaries with random contiguous subsequences of samples from the training set, with a length equal to two thirds of the original sample length. When using the kSVD algorithm (which requires atoms of the same size as the samples), we simply used consistent positional zero padding.

Moreover, we kept the same methodological choices as in the original paper, which means we learned one dictionary per class on each dataset, using sparsity levels of 2, 5 and 10. We also used a number of iterations for the kSVD and TWI-kSVD algorithms that were consistent with the claims of the paper, which led us to use 10 iterations for each dataset. In order to fully use the time-warp invariance of the TWI-OMP algorithm, we did not restrict the window size when computing the optimal alignment between the atoms and the samples, at the cost of very long runtimes.

4 Results

4.1 BME Dataset

First, let's have a look at a comparative example on Figure 3, where the same sample, has been reconstructed by both algorithms. There, we can see that the TWI-kSVD algorithm is able to reconstruct the sample with a lower reconstruction error at each sparsity level.

When we look at the reconstruction errors of both algorithms on the BME dataset in Table 1, we can see that our implementation of the TWI-kSVD algorithm is able to reproduce the results of the original paper to some extent, in the sense that it gives a lower reconstruction error than the kSVD algorithm for all sparsity levels. However, the classification errors in Table 2 show that our implementation of kSVD gives a lower classification error than expected, whereas TWI-kSVD gives a classification error that is way higher than in the original paper, no matter the classification strategy. This could be due to the assumptions we made regarding the preprocessing of the dataset and the initialization of the dictionaries, but after trying many different approaches (no subsampling, different initialization strategies, etc...), we have not been able to reproduce the results of the original paper, which led us to question the reproducibility of the results. Contacting the authors also did not help, as we did not receive any answer.

Sparsity/Models	kSVD	TWI-kSVD
2	0.25551	0.17313
5	0.19873	0.12301
10	**0.15060**	**0.08559**

Table 1: Reconstruction L_2 errors on the BME dataset (lower is better)

This is confirmed by the analysis of the reconstruction loss presented in Figure 4. Indeed, we can see that the reconstruction loss of the TWI-kSVD algorithm is not decreasing as fast as the one of the kSVD algorithm, which might explain the higher classification error. We also tried to visualize the learned dictionaries as shown in Figure 5.

We also observe in Table 2 that our classification strategy provides a much lower classification error than the original one for the kSVD algorithm, but not for the TWI-kSVD algorithm. Our interpretation for the performance gap of the TWI-kSVD between our implementation and the original paper is that both classification criteria are based on the fact that the error reconstruction is lower for the correct class than for the other classes. However, as we can see in Figure 8, the average reconstruction error is not the same for each class, which might explain the higher classification error.

Sparsity/Models	kSVD	TWI-kSVD
2	**0.22**	0.58
5	0.30	0.58
10	0.45	**0.47**

Original classification strategy

Sparsity/Models	kSVD	TWI-kSVD
2	**0.07**	**0.53**
5	0.08	**0.53**
10	0.10	0.62

Our classification strategy

Table 2: Classification error rates on the BME dataset (lower is better)

4.2 DIGITS Dataset

Now, let's have a look at a comparative example on Figure 6, where the same sample is reconstructed by both algorithms. We see that TWI-kSVD is able to reconstruct the sample with a lower sparsity level than kSVD, and overall with a lower reconstruction error.

Table 3 shows a higher reconstruction error with TWI-kSVD, which is the opposite of what was observed on the BME dataset and claimed in the original paper. We think this comes from the complexity of the dataset, and that TWI-kSVD is not able to learn a good dictionary, as shown in Figure 7.

Sparsity/Models	kSVD	TWI-kSVD
2	0.50133	0.53786
5	0.30436	0.32883
10	**0.20911**	**0.21764**

Table 3: Reconstruction L_2 errors on the DIGITS dataset (lower is better)

Regarding the classification errors, we still observe a great performance gap between our implementation and the original paper, as shown in Table 4. However, we observe that our classification strategy provides a lower classification error, at least for the kSVD algorithm.

Sparsity/Models	kSVD	TWI-kSVD
2	**0.27**	0.73
5	0.53	**0.66**
10	0.63	0.92

Original classification strategy

Sparsity/Models	kSVD	TWI-kSVD
2	**0.14**	0.81
5	0.27	**0.76**
10	0.29	0.88

Our classification strategy

Table 4: Classification error rates on the DIGITS dataset (lower is better)

References

[1] MALLAT, S. G., AND ZHIFENG, Z. Matching pursuits with time-frequency dictionnaries.

[2] MICHAL, A., MICHAEL, E., AND ALFRED, B. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation.

[3] PATI, Y. C., R., R., AND P.S, K. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition.

[4] SAEED, V. Y. Time warp invariant sparse coding and dictionary learning for time series classification and clustering.

[5] SAEED, V. Y., AND DOUZAL-CHOUAKRIAA, A. Time warp invariant ksvd: Sparse coding and dictionary learning for time series under time warp.

5 Appendix

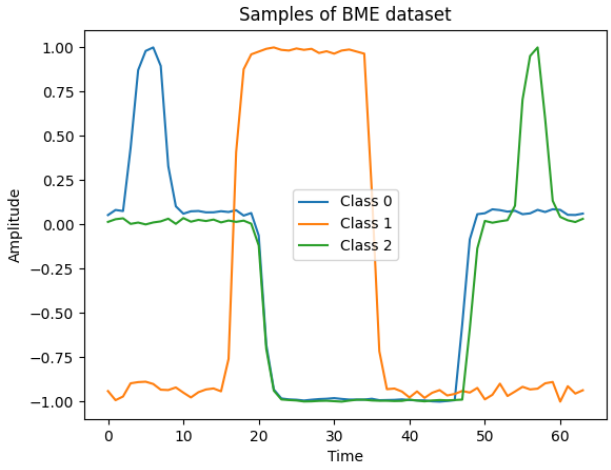


Figure 1: Samples of each class of the BME dataset

Algorithm 1 TWI-OMP algorithm

Require: $\boldsymbol{x}, \boldsymbol{D}, \tau$
  $\boldsymbol{r} = \boldsymbol{x}, \Omega = \emptyset$
  for $t \in \{1, ..., \tau\}$ **do**
    For all $k \notin \Omega$, compute $\boldsymbol{\Delta}_k$ and $C_k = \text{cosine}(\boldsymbol{x}, \boldsymbol{d_k})$ using dynamic programming
    $j = \arg\max_{k \notin \Omega} C_k$
    $\Omega = \Omega \cup \{j\}$, $\boldsymbol{S}_\Omega = (\boldsymbol{d}_j^{\boldsymbol{s}})_{j \in \Omega}$
    $\boldsymbol{\alpha}_\Omega = \arg\min_{\boldsymbol{\alpha}} \|\boldsymbol{x} - \boldsymbol{S}_\Omega \boldsymbol{\alpha}\|_2$
    $\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{S}_\Omega \boldsymbol{\alpha}_\Omega$
  end for
  return $\boldsymbol{\alpha}_\Omega, \boldsymbol{\Delta} = (\boldsymbol{\Delta_k})_{k \in \Omega}$

Algorithm 2 TWI-kSVD algorithm

Require: $\boldsymbol{X}, \boldsymbol{D}, \tau$
  for $t \in \{1, ..., n_{\text{iter}}\}$ **do**
    for $i \in \{1, ..., N\}$ **do**
      $\boldsymbol{\alpha}_i, (\boldsymbol{\Delta_{i,k}})_{k=1}^K = \text{TWI\_OMP}(\boldsymbol{x}_i, \boldsymbol{D}, \tau)$
    end for
    for $k \in \{1, ..., K\}$ **do**
      $\omega_k = \{i \in \{1, ..., N\} : \boldsymbol{\alpha}_{i,k} \neq 0\}$
      $\boldsymbol{E_k} = [\boldsymbol{e}_i]_{i \in \omega_k}$
      $\tilde{\boldsymbol{E}}_{\boldsymbol{k}} = [\varphi(\boldsymbol{e}_i)]_{i \in \omega_k}$
      $\boldsymbol{E_k} = \boldsymbol{U\Sigma V^T}$
      $\boldsymbol{d_k} = \boldsymbol{u}_1$
      for $i \in \omega_k$ **do**
        $\gamma_i(\boldsymbol{u}_1) = \boldsymbol{\Delta}_{i,k}\, \text{rotation}(\boldsymbol{u}_1, \boldsymbol{d}_k, \boldsymbol{\Delta}_{i,k}^T \boldsymbol{d}_k^{s_i})$
        $\boldsymbol{d}_k^{s_i} = \gamma_i(\boldsymbol{u}_1)$
        $\boldsymbol{\alpha}_{i,k} = \dfrac{\boldsymbol{e}_i^T \boldsymbol{\gamma_i(u_1)}}{\|\boldsymbol{\gamma_i(u_1)}\|}$
      end for
    end for
  end for
  return $A, \boldsymbol{\Delta}, \boldsymbol{D}$

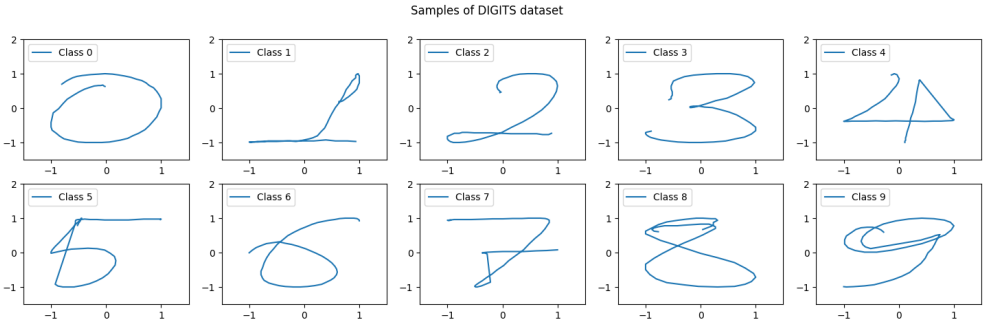
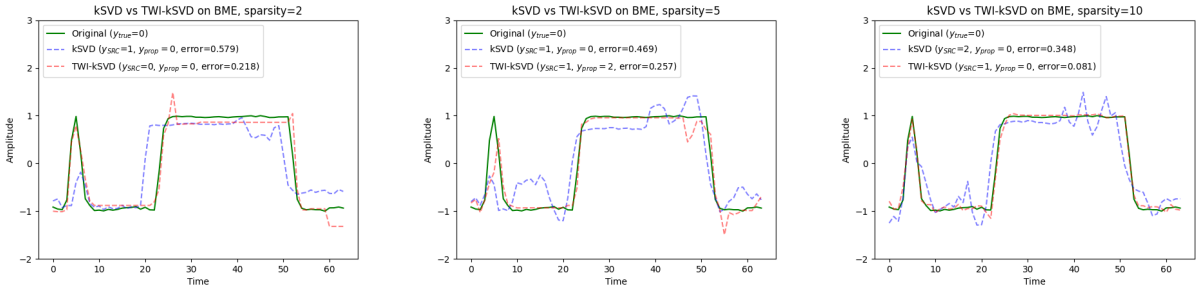
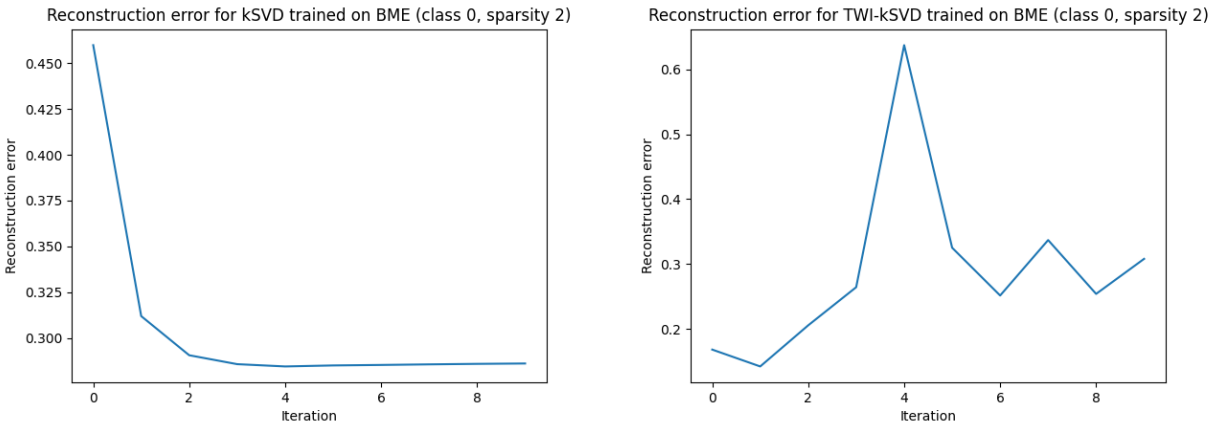


Figure 2: Samples of each class of the DIGITS dataset



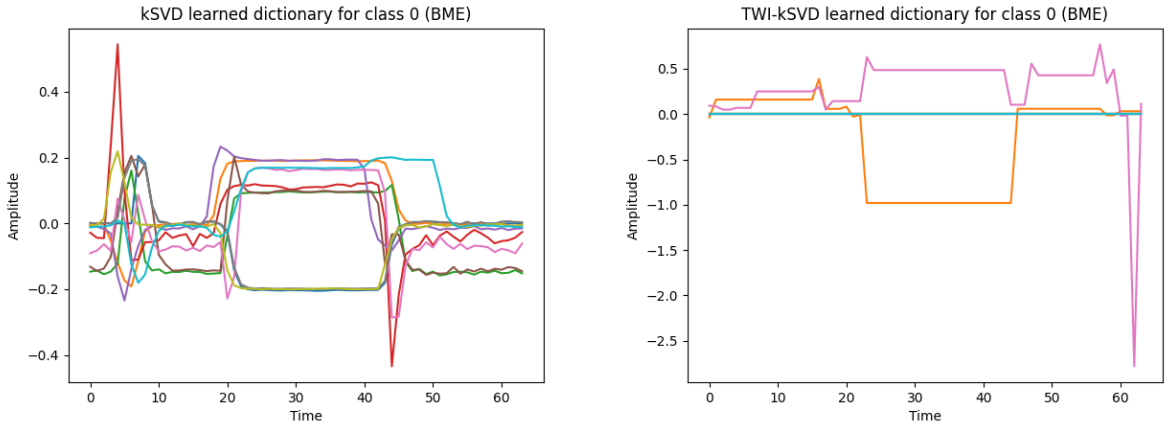
Sparsity = 2 Sparsity = 5 Sparsity = 10

Figure 3: Example of reconstructions (BME dataset) with different sparsity levels



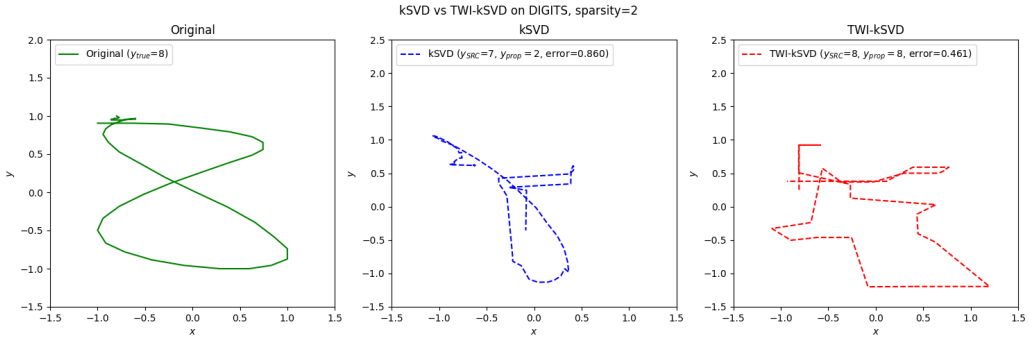
(a) kSVD (b) TWI-kSVD

Figure 4: Evolution of reconstruction loss during training (BME dataset)

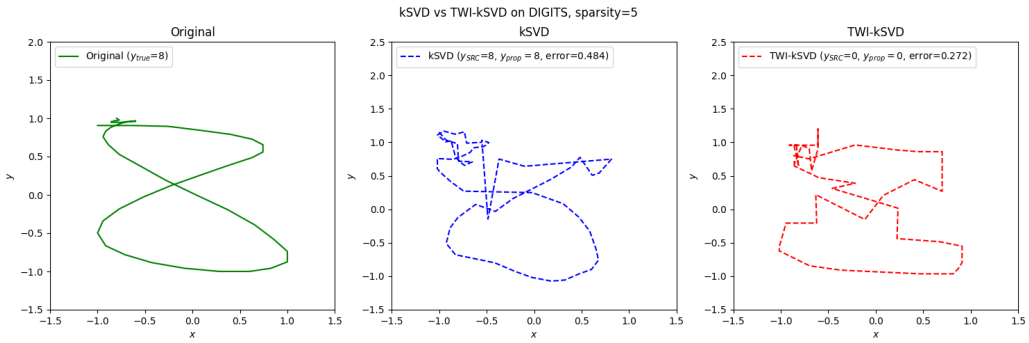


kSVD TWI-kSVD

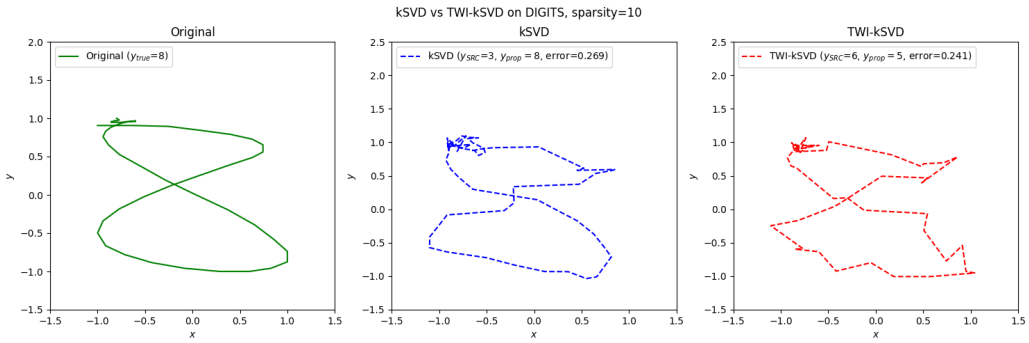
Figure 5: Learned dictionaries for first class of BME dataset



Sparsity = 2

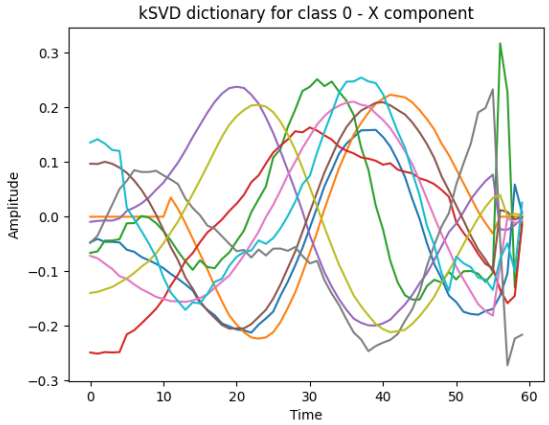


Sparsity = 5

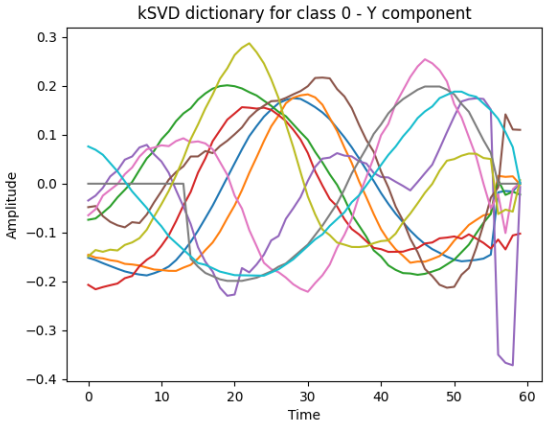


Sparsity = 10

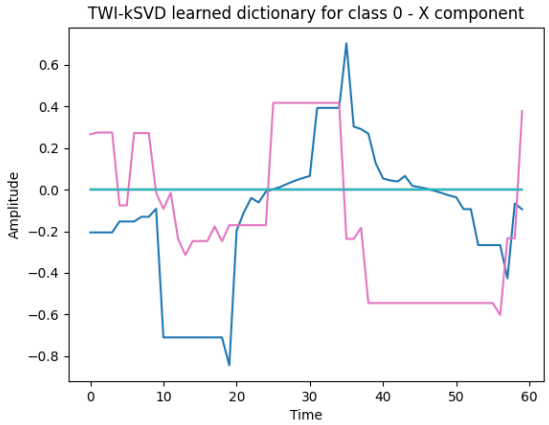
Figure 6: Example of reconstructions (DIGITS dataset) with different sparsity levels



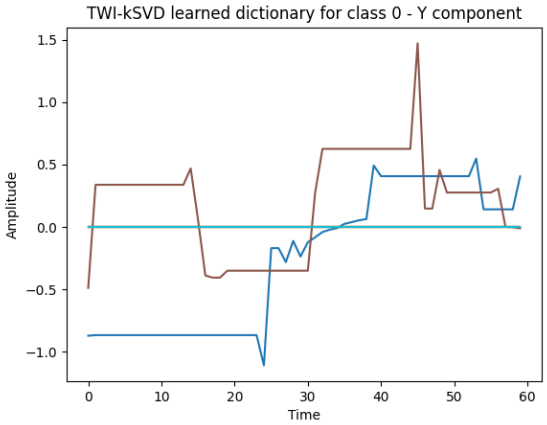
kSVD - X component



kSVD - Y component

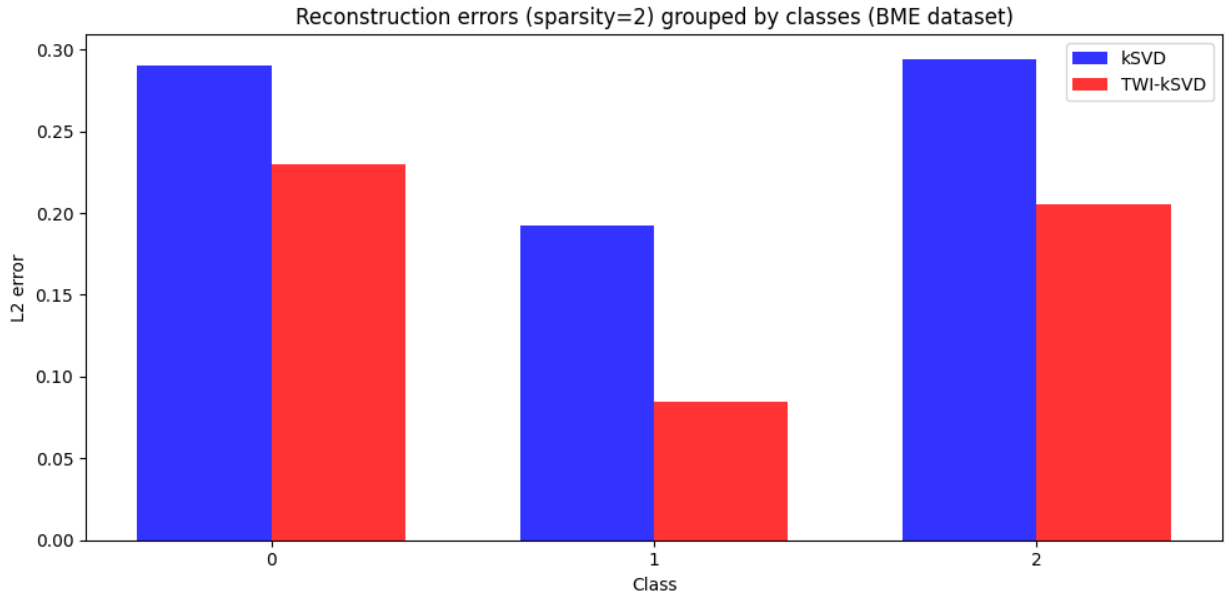


TWI-kSVD - X component



TWI-kSVD - Y component

Figure 7: Learned dictionaries for first class of DIGITS dataset



BME

DIGITS

Figure 8: Reconstruction errors grouped by class (both datasets, sparsity = 2)