A Novel Kernel Sparse Coding Method with A Two-stage Acceleration Strategy

Anonymous authors
Paper under double-blind review

Abstract

Sparse coding aims to exploit the latent linear structure of the input data, transforming dense data into sparse data, thereby improving data processing efficiency. However, many real-word signals cannot be expressed linearly, rendering the traditional sparse coding algorithms ineffective. One potential solution is to expand the dimensions of data. In this paper, we verify that the feature mapping of Radial Basis Function (RBF) kernel contains infinite dimensional information, and it does not significantly increase the computational complexity. Based on this, we propose to explore the l_1-norm regularization sparse coding method with RBF kernel, and provides a solution with convergence guarantees by leveraging the principle of coordinate descent. Additionally, to accelerate the optimization process, we introduce a novel two-stage acceleration strategy, based on theoretical analysis and empirical observations. Experimental results demonstrate that the two-stage acceleration strategy can reduce processing time by up to 90%. Furthermore, when the data size is compressed to about 2% of its original scale, the NMAE metric of the proposed method reaches as low as 0.0824 to 0.2195, achieving a significant improvement of up to 47% compared to traditional linear sparse coding methods and 36% compared to other kernel sparse coding techniques.

1 Introduction

Sparse coding, known for its efficiency in transforming data into a sparse and interpretable format, posits that data in the input space possesses inherent features (Pati et al., 1993; Bao et al., 2014; Lee et al., 2006; Kim, 2014). Within mathematical models, these intrinsic features are typically modeled as a set of basic atoms or dictionary atoms. As illustrated in Fig. 1, these atoms can be concatenated into a dictionary matrix $\mathbf{X}$. Once the input data $\mathbf{y}$ can be expressed as a linear combination of a few dictionary atoms (e.g., Atom 2 and Atom 6), it can be encoded as sparse weights data $\mathbf{w}$, where the majority of weights (e.g., w_j for $j = 1, 3, 4, 5, 7, 8$) are zeros and only a few weights (e.g., w_2, w_6) are nonzeros. Owing to the unique advantages of sparse data, sparse coding have the potential capability to handle the issues of data compression (Ramabadran & Sinha, 1994; Watkins et al., 2018; Li et al., 2016; Wang et al., 2016), efficient computation (Schütze et al., 2016; Chalk et al., 2018; Bengio et al., 2009; Schütze et al., 2016), feature analysis (Yutani et al., 2022; Shi et al., 2021; Tong et al., 2019), denoising (Liu et al., 2019; Wang et al., 2020; Lu et al., 2015).

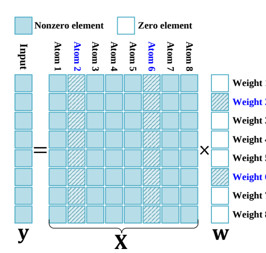


Figure 1: Sparse coding.

Within the aforementioned sparse coding framework, the input data $\mathbf{y}$ can be sparsely encoded into $\mathbf{w}$ because it can be linearly reconstructed by a combination of several dictionary atoms. It is worth noting that the dictionary atoms are typically learned directly from the input space, leveraging the fundamental assumption that the input data possesses inherent linear structures. However, (Yang et al., 2016) has shown that many practical signals do not possess this characteristic, limiting the effectiveness of traditional linear sparse coding methods in such scenarios. Therefore, seeking methods to effectively address the sparse coding of complex data is a highly meaningful topic.

Motivations. Recent studies show that a transformation of low-dimensional data into a higher-dimensional space, may enable data that is inherently nonlinearly separable in its original form to become linearly distinguishable within the expanded feature space (Schölkopf et al., 1997). For example, as shown in Fig. 2(a), there is no straight line that can separate the green data points from the red data points. However, if we map each two dimensional data point (x, y) into a three dimensional data point (x, y, xy), as shown in Fig. 2(b), there will be numerous planes that can separate the green data points from the red data points. Inspired by this, increasing the dimensionality of input data may offer promising prospects for sparse coding problems involving data with complex internal structures.

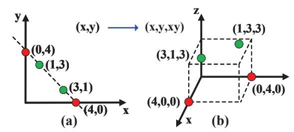


Figure 2: Transforming linearly inseparable data into linearly separable data.

While increasing the dimensionality of input data can enhance its discriminability, facilitating sparse coding analysis, it also inevitably introduce problems, including indeterminate forms and scales of data dimensionality enhancement, and high computational complexity. It is known that kernel trick provides an effective way to expand data dimensions with reducing computational complexity, and there are various categories of kernel functions (Gao et al., 2010). To select a good kernel function, we must be aware that the richer the dimensional information contained in the feature map, the better the performance of the corresponding kernel function. Therefore, this article delves into the specific form of feature mapping corresponding to the Radial Basis Function (RBF) kernel and analyzes its potential to demonstrate great performance. Ultimately, we solidify our decision to employ the RBF kernel-based approach as a means to tackle the challenge of enhancing sparse coding efficiency.

Contributions. The main contributions of this work include:

• We verify that the feature mapping of the Radial Basis Function (RBF) kernel contains infinite dimensional information, and it does not significantly increase the computational complexity. Based on this, we explore the l_1-norm regularized sparse coding method with RBF kernel, and provides a solution with convergence guarantees by leveraging the principle of coordinate descent.

• To reduce the computational complexity of the standard kernel sparse coding method based on coordinate descent, a novel two-stage acceleration strategy is introduced. This strategy focuses on updating the initially predicted nonzero weight factors, and effectively predicts zero weight factors to skip the computation of the corresponding intermediate variables.

• Our experiments demonstrate that the proposed two-stage acceleration strategy significantly reduces processing time by up to 90%. Additionally, our method outperforms both traditional linear sparse coding and other kernel sparse coding methods. Specifically, when the data size is compressed to about 2% of its original scale, the NMAE metric of the proposed method reaches as low as 0.0824 to 0.2195, achieving a significant improvement of up to 47% compared to traditional linear sparse coding methods and 36% compared to other kernel sparse coding techniques.

2 Preliminary and kernel trick

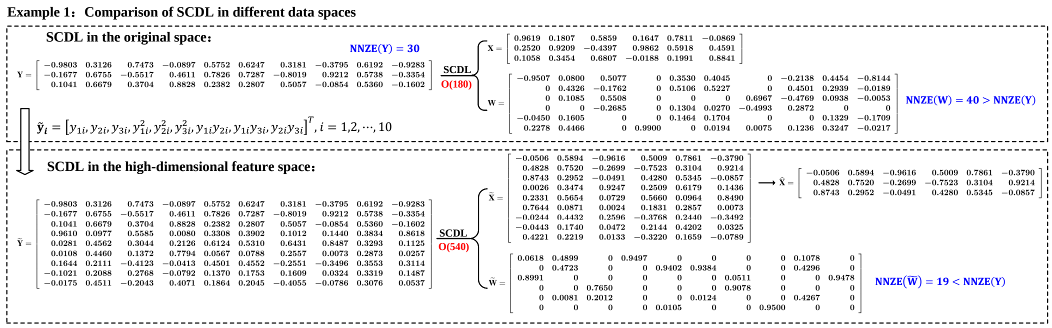
Problem: In practice, the effective implementation of sparse coding relies on a dictionary matrix $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_p] \in R^{n \times p}$ that preserves the potential features of the input space, which should be learned in advance from the dense historical data $\mathbf{Y} \in R^{n \times m}$ as:

$$\min_{\mathbf{X},\mathbf{W}} \frac{1}{2}\|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda \sum_{j=1}^{m} \|\mathbf{w}_j\|_1, \tag{1}$$

where $\mathbf{W} \in R^{p \times m}$ is the sparse coding matrix, with the majority of elements being zero, and only a minority of elements being nonzero. $\mathbf{w}_j$ is the jth column of $\mathbf{W}$. The first term for (1) is the approximation error, and the second term with penalty parameter λ is used to force the columns of matrix $\mathbf{W}$ to be sparse.

Essentially, optimization problem (1) focuses on identifying a dictionary matrix $\mathbf{X}$ within the original data space, enabling the transformation of $\mathbf{Y}$ into a sparse matrix $\mathbf{W}$, based on the assumption that each column in $\mathbf{Y}$ can be expressed as a linear combination of a few atoms in $\mathbf{X}$. Since $\mathbf{W}$ contains significantly less amount of data than $\mathbf{Y}$, it can be seen as a sparse form of input data $\mathbf{Y}$. **However, when the input data $\mathbf{Y}$ does not have enough inherent linear structure, achieving a sparse linear approximation of $\mathbf{Y}$ with high precision proves challenging.**

For simplicity, we abbreviate "Number of Nonzero Elements" as "NNZE" in this paper. As shown in Example 1, the input data $\mathbf{Y}$ contains 10 data points uniformly and randomly sampled from the unit sphere. Assuming the dictionary matrix has 6 atoms, performing sparse coding and dictionary learning on $\mathbf{Y}$ in the original input space, with the constraint that the approximation error



$AE_{original} = \frac{1}{2}\|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_F^2 < 0.1$, yields the dictionary matrix $\mathbf{X}$ and the sparse matrix $\mathbf{W}$. In this case, $\mathbf{W}$ includes 40 nonzero elements (i.e., NNZE($\mathbf{W}$)=40), which exceeds the amount of data in the original $\mathbf{Y}$ (NNZE($\mathbf{Y}$)=30). Therefore, $\mathbf{W}$ cannot be considered as a sparse form of $\mathbf{Y}$.

Elevating Data Dimensions: Inspired by the example in Fig. 2, elevating the dimensionality of data may potentially enhance the effectiveness of sparse coding. In Example 1, we directly extend each $\mathbf{y}_i = [y_{1i}, y_{2i}, y_{3i}]^T$ vertically to $\widetilde{\mathbf{y}}_i = [y_{1i}, y_{2i}, y_{3i}, y_{1i}^2, y_{2i}^2, y_{3i}^2, y_{1i}y_{2i}, y_{1i}y_{3i}, y_{2i}y_{3i}]^T$, for $i = 1, 2, \cdots, 10$, forming a high-dimensional data matrix $\widetilde{\mathbf{Y}}$. Given that the dictionary matrix is composed of 6 atoms, conducting sparse coding and dictionary learning on $\widetilde{\mathbf{Y}}$ in the high-dimensional feature space, with the constraint that the approximation error $AE_{original} = \frac{1}{2}\|\mathbf{Y} - \widehat{\mathbf{X}}\widetilde{\mathbf{W}}\|_F^2 < 0.1$, yields the dictionary matrix $\widetilde{\mathbf{X}}$ and the sparse matrix $\widetilde{\mathbf{W}}$. Note that, just as $\widetilde{\mathbf{Y}}$ is a vertical expansion of $\mathbf{Y}$, $\widetilde{\mathbf{X}}$ is a vertical expansion of $\widehat{\mathbf{X}}$. In this case, there are only 19 nonzero elements in $\widetilde{\mathbf{W}}$ (i.e., NNZE($\widetilde{\mathbf{W}}$)=19), which is less than that of the original data $\mathbf{Y}$ (NNZE($\mathbf{Y}$)=30). Consequently, $\widetilde{\mathbf{W}}$ can be regarded as a sparse representation of $\mathbf{Y}$. That is, **expanding dimensions can indeed result in a more efficient sparse representation of the original data**.

Indeed, there are numerous nonlinear mapping methods that can be utilized to enhance the dimensionality of data. For instance, similar to the approach in Example 1 where all second-order terms of the original data are introduced, we could also incorporate all third-order terms, all fourth-order terms, and so on. Additionally, we could introduce a combination of all second-order and third-order terms, or selectively introduce parts of second-order and third-order terms, and so forth. In essence, the ways to increase dimensionality are virtually limitless, and it is impractical to verify each method individually to determine which best meets practical needs. Therefore, **the specific form and scale of data dimensionality enhancement that can enhance the efficiency of sparse coding remains a thought-provoking issue**.

Theoretically, the more nonlinear terms that are incorporated, the more likely it is to enhance the efficiency of sparse coding. However, **as the dimensionality of the data increases, the computational complexity inevitably rises as well**. As shown in Example 1, the computational complexity of sparsely coding $\mathbf{Y}$ into $\mathbf{X}$ and $\mathbf{W}$ is $O(180)$. After expanding $\mathbf{Y}$ into $\widetilde{\mathbf{Y}}$, the computational complexity of sparsely coding $\widetilde{\mathbf{Y}}$ into $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{W}}$ increases to $O(540)$. If the scale of $\mathbf{Y}$ is too large or if the data expansion method is too complex, the corresponding computational load may become prohibitive. Therefore, overcoming the computational complexity introduced by data dimensionality expansion remains a significant challenge.

Advantages of RBF Kernel in Elevating Data Dimensions: In response to the aforementioned challenges of "indeterminate forms and scales, and high computational complexity of data dimensionality enhancement", in this section, we validate that the Radial Basis Function (RBF) kernel possesses excellent properties that give it distinct advantages in elevating data dimensions.

For any $\mathbf{a}, \mathbf{b} \in R^n$, their RBF kernel $\kappa(\mathbf{a}, \mathbf{b}) = \exp(-\frac{1}{2\sigma^2}\|\mathbf{a} - \mathbf{b}\|_2^2)$ can be decomposed as

$$\kappa(\mathbf{a}, \mathbf{b}) = \exp\left(-\frac{1}{2\sigma^2}\left(\|\mathbf{a}\|_2^2 + \|\mathbf{b}\|_2^2\right)\right)\exp\left(\frac{\mathbf{a}^T\mathbf{b}}{\sigma^2}\right) = \phi(\mathbf{a})^T\phi(\mathbf{b}) \tag{2}$$

where σ is the RBF kernel hyper-parameter, $\phi(\mathbf{z}) = \left[\sqrt{\frac{\varrho}{\sigma^{2q}q_1!\cdots q_n!}}(z_1^{q_1}\cdots z_n^{q_n})\right]_{|\mathbf{q}|=0}^{\infty}$ is an infinite order polynomial map for $\mathbf{z} = \mathbf{a}$ or $\mathbf{b}$, $|\mathbf{q}| = q_1 + \cdots + q_n$, and $\varrho = \exp\left(-\frac{1}{2\sigma^2}\left(\|\mathbf{a}\|_2^2 + \|\mathbf{b}\|_2^2\right)\right)$.

Notably, on the one hand, $\phi(\mathbf{z})$ includes all integer-order terms of the original data $\mathbf{z}$, including infinite order. That is to say, $\phi(\mathbf{z})$ **contains all forms of data expansion, and the scale of the expanded data is infinite**. On the other hand, the inner product of two expanded data points $\phi(\mathbf{a})$ and $\phi(\mathbf{b})$ can be directly calculated through kernel function $\kappa(\mathbf{a}, \mathbf{b})$ with computation complexity of $O(n)$, instead of $\phi(\mathbf{a})^T\phi(\mathbf{b})$ with computation complexity of ∞. That is to say, in the calculation process, **as long as the inner product of extended data points is formed, the problem of computational complexity resulting from dimensionality increase can be avoided via kernel tricks**. Precisely because of these great properties, the RBF kernel is expected to exhibit excellent performance in expanding data dimensions. Consequently, our endeavor in this paper revolves around leveraging the feature mapping ϕ, inherent to the RBF kernel, to expand data dimensions, ultimately enhanceing the performance of sparse coding.

3 Kernel sparse coding dictionary learning

Based on the discussion above, in order to achieve a more efficient sparse coding scheme for the input data $\mathbf{Y}$, we impose the feature mapping in (2) to the historical data $\mathbf{Y}$ and the dictionary matrix $\mathbf{X}$, formulating the following Kernel Sparse Coding Dictionary Learning (KSCDL) problem:

$$\min_{\mathbf{X},\mathbf{W}} \frac{1}{2}\|\phi(\mathbf{Y}) - \phi(\mathbf{X})\mathbf{W}\|_F^2 + \lambda \sum_{j=1}^{m} \|\mathbf{w}_j\|_1. \tag{3}$$

It is noteworthy that, although the KSCDL in (3) is an extension of (1), the solution of (3) can not be obtained by extending the solution of (1). For an in-depth discussion, please refer to A.3.

Solution Overview: To solve KSCDL (3), we decompose the KSCDL problem into a Kernel Sparse Coding (KSC) subproblem and a Kernel Dictionary Learning (KDL) subproblem:

• **KSC:** The KSC subproblem of (3) can be interpreted as finding the weight matrix $\mathbf{W}$, while keeping the dictionary matrix $\mathbf{X}$ fixed, through

$$\mathbf{W} = \arg\min_{\mathbf{W}} \frac{1}{2} \sum_{j=1}^{m} \left[\|\phi(\mathbf{y}_j) - \phi(\mathbf{X})\mathbf{w}_j\|_2^2 + \lambda \|\mathbf{w}_j\|_1 \right]. \tag{4}$$

• **KDL:** The KDL subproblem of (3) involves solving the dictionary matrix $\mathbf{X}$, with the sparse coding matrix $\mathbf{W}$ fixed, through

$$\mathbf{X} = \arg\min_{\mathbf{X}} \frac{1}{2}\|\phi(\mathbf{Y}) - \phi(\mathbf{X})\mathbf{W}\|_F^2. \tag{5}$$

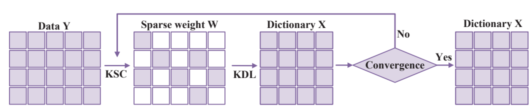


Figure 3: Flowchart of KSCDL.

Moving forward, based on the block coordinate descent method, we alternately and iteratively solve these two sub-problems as shown in Fig. 3. Upon convergence of the algorithm, we output and save the dictionary matrix $\mathbf{X}$.

Solution for KSC: For each column of $\mathbf{W}$ and $\mathbf{Y}$ (abbreviated as $\mathbf{w}$ and $\mathbf{y}$, respectively), the objective function in (4) can be equivalent rewritten as:

$$\min_{\mathbf{w}} \frac{1}{2}\|\phi(\mathbf{y}) - \phi(\mathbf{X})\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_1, \tag{6}$$

Based on the kernel trick, we have $\|\phi(\mathbf{y}) - \phi(\mathbf{X})\mathbf{w}\|_2^2 = \kappa_{\mathbf{yy}} - 2\kappa_{\mathbf{yX}}\mathbf{w} + \mathbf{w}^T\kappa_{\mathbf{XX}}\mathbf{w}$. In which $\kappa_{\mathbf{yy}} = \kappa(\mathbf{y}, \mathbf{y}) = \phi(\mathbf{y})^T\phi(\mathbf{y})$, $\kappa_{\mathbf{yX}} = \kappa(\mathbf{y}, \mathbf{X}) = \phi(\mathbf{y})^T\phi(\mathbf{X}) = [\kappa(\mathbf{y}, \mathbf{x}_1), \cdots, \kappa(\mathbf{y}, \mathbf{x}_p)]$, and $\kappa_{\mathbf{XX}} = \phi(\mathbf{X})^T\phi(\mathbf{X})$ with $[\kappa(\mathbf{X}, \mathbf{X})]_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Then problem (6) is equivalent to

$$\min_{\mathbf{w}} \; -\kappa_{\mathbf{yX}}\mathbf{w} + \frac{1}{2}\mathbf{w}^T\kappa_{\mathbf{XX}}\mathbf{w} + \lambda\|\mathbf{w}\|_1. \tag{7}$$

Denote the objective function of problem (7) as $\mathfrak{J}$, if $w_i > 0$, the derivative of $\mathfrak{J}$ with respect to w_i can be obtained as $\frac{\partial \mathfrak{J}}{\partial w_i} = -\kappa_{\mathbf{yx}_i} + \sum_{j=1}^{p} \kappa_{ij} w_j + \lambda$, where $\kappa_{\mathbf{yx}_i} = \kappa(\mathbf{y}, \mathbf{x}_i)$ and $\kappa_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Let $\frac{\partial \mathfrak{J}}{\partial w_i} = 0$, one gets $-\kappa_{\mathbf{yx}_i} + \sum_{j\neq i}^{p} \kappa_{ij} w_j + \kappa_{ii} w_i + \lambda = 0$. That is $w_i = \kappa_{\mathbf{yx}_i} - \sum_{j\neq i}^{p} \kappa_{ij} w_j - \lambda$, since $\kappa_{ii} = 1$. For convenience, we denote $z_i = \kappa_{\mathbf{yx}_i} - \sum_{j\neq i}^{p} \kappa_{ij} w_j$. To avoid confusion, we refer to one complete iteration of the coordinate descent algorithm as a round, which starts from updating

w_1 and continues until w_p has been updated. The variables updated in the kth round will be marked with a superscript (k), then

$$z_i^{(k)} = \kappa_{\mathbf{yx}_i} - \sum_{j\neq i}^{p} \kappa_{ij} w_j^{[i,k]}, \text{ where } w_j^{[i,k]} = \begin{cases} w_j^{(k)}, & \text{if } j < i; \\ w_j^{(k-1)}, & \text{if } j > i. \end{cases} \tag{8}$$

By analogy, it's not difficult to summarize that

$$w_i^{(k)} = \mathrm{soft}_\lambda(z_i^{(k)}) = \begin{cases} z_i^{(k)} - \lambda, & \text{if } z_i^{(k)} > \lambda; \\ z_i^{(k)} + \lambda, & \text{if } z_i^{(k)} < -\lambda; \\ 0, & \text{if } |z_i^{(k)}| \leq \lambda. \end{cases} \tag{9}$$

Solution for KDL: Based on the kernel trick, the objective function in (5) can be rewritten as

$$\|\phi(\mathbf{Y}) - \phi(\mathbf{X})\mathbf{W}^{(k)}\|_F^2 = \mathrm{Trace}\Big[\kappa_{\mathbf{YY}} - 2\kappa_{\mathbf{YX}}\mathbf{W}^{(k)} + (\mathbf{W}^{(k)})^T \kappa_{\mathbf{XX}}\mathbf{W}^{(k)}\Big]. \tag{10}$$

where $\kappa_{\mathbf{YY}} = \kappa(\mathbf{Y},\mathbf{Y}) = \phi(\mathbf{Y})^T\phi(\mathbf{Y})$, $\kappa_{\mathbf{YX}} = \kappa(\mathbf{Y},\mathbf{X}) = \phi(\mathbf{Y})^T\phi(\mathbf{X})$. There is no closed form solution for dictionary matrix $\mathbf{X}$ due to the kernel matrices, we update $\mathbf{X}$ via the gradient descent method. The gradient of the objective function w.r.t. matrix $\mathbf{X}$ can be calculated as $G_{\mathbf{X}} = \frac{1}{\sigma^2}(\mathbf{X}\mathbf{\Gamma}_1 - \mathbf{Y}\mathbf{Q}_1 + \mathbf{X}\mathbf{Q}_2 - \mathbf{X}\mathbf{\Gamma}_2)$. Here $\mathbf{Q}_1 = (\mathbf{W}^{(k)})^T \odot \kappa_{\mathbf{YX}}$, $\mathbf{\Gamma}_1 = \mathrm{diag}(\mathbf{1}_n^T\mathbf{Q}_1)$, $\mathbf{Q}_2 = \mathbf{W}^{(k)}(\mathbf{W}^{(k)})^T \odot \kappa_{\mathbf{XX}}$, $\mathbf{\Gamma}_2 = \mathrm{diag}(\mathbf{1}_r^T\mathbf{Q}_2)$. We update $\mathbf{X}$ as

$$\mathbf{X}^{(k)} \leftarrow \mathbf{X}^{(k-1)} - \mu G_{\mathbf{X}}, \tag{11}$$

where μ is the optimization stepsize. As outlined in Fig.3, the organization details of the KSCDL algorithm and the theoretical guarantee of its convergence are provided in A.2.

4 Fast kernel sparse coding

The KSC subproblem is solved basing on the coordinate descent principle, which guarantees convergence. However, the updating iterative process of the weight vector is slow.

Computational complexity analysis. In Fig. 4, we provide a illustration of the update process for sparse weight vector $\mathbf{w}$ during the kth round of iteration. It can be observed that the calculation of the intermediate variable $z_i^{(k)} = \kappa_{\mathbf{yx}_i} - \sum_{j\neq i}^{p} \kappa_{ij} w_j^{[i,k]}$, for each sparse weight factor $w_i^{(k)}$, has a complexity of $O(p)$, and there are p such weight factors (i.e., $w_i^{(k)}, i = 1, 2, \cdots, p$) in each weight vector $\mathbf{w}$. Therefore, the computational complexity for each iteration of $\mathbf{w}$ is $O(p^2)$, making the update process relatively sluggish. In the following, we will approach from two aspects: theoretical analysis and experimental observation, to accelerate the KSC process in two stages.

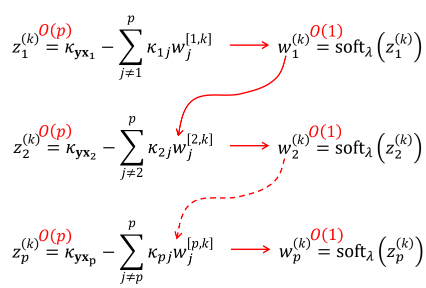


Figure 4: Standard updating process for sparse weight vector $\mathbf{w}$.

4.1 Acceleration Strategy I: safely skipping updates for zero elements

Theoretical Analysis: Equation (9) shows that $w_i^{(k)}$ is essentially a soft threshold function of $z_i^{(k)}$ with the threshold λ. It is particularly important to note that if $|z_i^{(k)}| \leq \lambda$ holds, we can directly obtain $w_i^{(k)} = 0$. In other words, there may be a scenario in which $w_i^{(k)}$ does not require an update calculation and can be set to 0 directly. In order to anticipate the relationship between $z_i^{(k)}$ and λ, we intend to explore the upper and lower bounds of $z_i^{(k)}$. For this purpose, we recompute $z_i^{(k)}$ as:

$$z_i^{(k)} = z_i^{(k-1)} - \kappa_{i:}\Delta\mathbf{w}_{(i)}^{(k)}, \tag{12}$$

where $\kappa_{i:} = [\kappa_{i1}, \kappa_{i2}, \cdots, \kappa_{ip}]$, $\Delta\mathbf{w}_{(i)}^{(k)} = [\Delta w_1^{(k)}, \cdots, \Delta w_{i-1}^{(k)}, 0, \Delta w_{i+1}^{(k-1)}, \cdots, \Delta w_p^{(k-1)}]^T$ and $\Delta w_l^{(k)} = w_l^{(k)} - w_l^{(k-1)}$ for each $l = 1, \cdots, p$. Combining with the Cauchy-Schwarz inequality (Steele, 2004), it can be obtained from (12) that

$$\underline{z}_i^{(k)} \leq z_i^{(k)} \leq \overline{z}_i^{(k)} \tag{13}$$

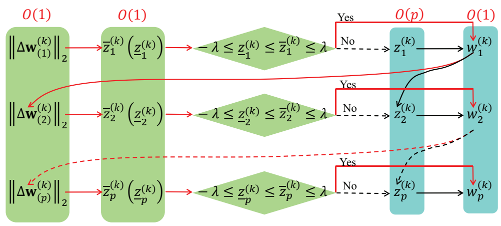


Figure 5: Acceleration Strategy I.

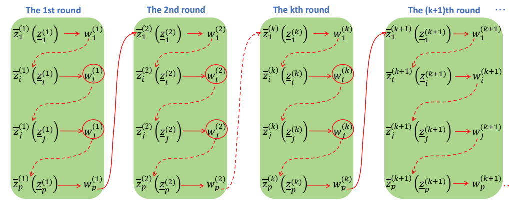


Figure 6: Complete update process of $\mathbf{w}$ based on Acceleration Strategy I.

where

$$\overline{z}_i^{(k)} = z_i^{(k-1)} + \|\kappa_{i:}\|_2 \|\Delta \mathbf{w}_{(i)}^{(k)}\|_2, \text{ and } \underline{z}_i^{(k)} = z_i^{(k-1)} - \|\kappa_{i:}\|_2 \|\Delta \mathbf{w}_{(i)}^{(k)}\|_2. \tag{14}$$

Obviously, based on $\overline{z}_i^{(k)}$, the computational complexity of $\underline{z}_i^{(k)}$ is $O(1)$. Naturally, we have

- If $\underline{z}_i^{(k)} > \lambda$ or $\overline{z}_i^{(k)} < -\lambda$, then $w_i^{(k)} \neq 0$, meaning w_i needs to undergo an update calculation.
- If $-\lambda \leq \underline{z}_i^{(k)} \leq \overline{z}_i^{(k)} \leq \lambda$, then $w_i^{(k)} = 0$, indicating that the update for w_i can be avoided.

The second scenario mentioned above indicates that there are indeed situations where it is not necessary to compute $z_i^{(k)}$, and only the calculations of $\overline{z}_i^{(k)}$ and $\underline{z}_i^{(k)}$ are needed to determine that $w_i^{(k)} = 0$, which can effectively reduce computational complexity.

In fact, $z_i^{(k-1)}$ can be directly obtained from the $(i-1)$th round and $\|\kappa_{i:}\|_2$ can be precomputed before iterations. In addition, based on the definition of $\Delta \mathbf{w}_{(i)}^{(k)}$ in equation (12), we can get

$$\|\Delta \mathbf{w}_{(i)}^{(k)}\|_2 = \sqrt{\|\Delta \mathbf{w}_{(i-1)}^{(k)}\|_2^2 - (\Delta w_i^{(k-1)})^2 + (\Delta w_{i-1}^{(k)})^2}. \tag{15}$$

That is, once we obtain $\|\Delta \mathbf{w}_{(i-1)}^{(k)}\|_2$, the computational complexity of $\|\Delta \mathbf{w}_{(i)}^{(k)}\|_2$ can be ignored. Therefore, for the kth round, the total computational complexity of $\overline{z}_i^{(k)}$ and $\underline{z}_i^{(k)}$, for all $i \in \{1, \cdots, p\}$, primarily arises from $\|\Delta \mathbf{w}_{(1)}^{(k)}\|_2^2$. Further more, we have

$$\|\Delta \mathbf{w}_{(1)}^{(k)}\|_2 = \sqrt{\|\Delta \mathbf{w}_{(p)}^{(k-1)}\|_2^2 - (\Delta w_1^{(k-1)})^2 + (\Delta w_p^{(k-1)})^2}. \tag{16}$$

That is, once we obtain $\|\Delta \mathbf{w}_{(p)}^{(k-1)}\|_2$, the computational complexity of $\|\Delta \mathbf{w}_{(1)}^{(k)}\|_2$ can be ignored. By analogy, for any $i \in \{1, \cdots, p\}$ and integer $k \geq 1$, the computational complexity of $\|\Delta \mathbf{w}_{(i)}^{(k)}\|_2$ primarily stems from $\|\Delta \mathbf{w}_{(1)}^{(1)}\|_2$, which is determined by the initialization rule of the algorithm. It can be 0 if we initialize $\Delta \mathbf{w}$ as a zero vector, or it can be $O(p)$ if we initialize $\Delta \mathbf{w}$ as a nonzero vector. From this perspective, using the upper and lower bounds $\overline{z}_i^{(k)}$ and $\underline{z}_i^{(k)}$ to anticipate zero elements in $\mathbf{w}$ holds the promise of reducing computational complexity.

Acceleration Strategy Design: Based on the above theoretical analysis, we propose to utilize $\overline{z}_i^{(k)}$ and $\underline{z}_i^{(k)}$ to accelerate the update of zero elements in the weight vector $\mathbf{w}$. As shown in Fig. 5, before calculating the intermediate variable $z_i^{(k)}$ for $w_i^{(k)}$, we first calculate the the upper and lower bounds of $z_i^{(k)}$ and use them to predict whether $w_i^{(k)}$ is zero. If it is, we directly skip the update of $z_i^{(k)}$; if not, we calculate $z_i^{(k)}$ and thereby update $w_i^{(k)}$. It is worth noting that **this acceleration strategy is not restricted to the case using RBF kernels, as $\mathbf{w}$ is calculated from $\kappa_{\mathbf{yX}}$ and $\kappa_{\mathbf{XX}}$, which are not constrained by any specific kernel function**.

4.2 ACCELERATION STRATEGY II: PRIORITIZING UPDATES FOR INITIALLY PREDICTED NONZERO ELEMENTS

Experimental Observation: Based on Acceleration Strategy I, if we update the elements of the weight vector $\mathbf{w}$ following the process shown in Fig. 6, and track the positions of nonzero elements

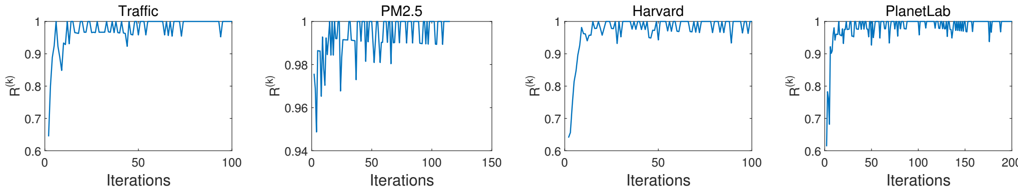


Figure 7: Changes for $R^{(k)}$ during the iteration process.

in $\mathbf{w}$, we will find that the positions of nonzero elements appear to be relatively stable. In other words, **the nonzero elements in vector $\mathbf{w}^{(k+1)}$ occupy positions that are closely analogous to those in vector $\mathbf{w}^{(k)}$**. To better describe this phenomenon, we define several new metrics to track the changes in the positions of nonzero elements throughout the iterative process:

- $\mathbb{I}^{(k)}$: the index set of nonzero elements in $\mathbf{w}^{(k)}$.
- $|\mathbb{I}^{(k)}|$: the number of nonzero elements in $\mathbf{w}^{(k)}$.
- $\mathbb{I}^{(k)} \cap \mathbb{I}^{(k+1)}$: the index set of elements that are nonzero in $\mathbf{w}^{(k)}$ and also nonzero in $\mathbf{w}^{(k+1)}$.
- $R^{(k)} = |\mathbb{I}^{(k)} \cap \mathbb{I}^{(k+1)}| \big/ |\mathbb{I}^{(k)}|$: the ratio of the number of elements that are nonzero in $\mathbf{w}^{(k)}$ and remain nonzero in $\mathbf{w}^{(k+1)}$, to the number of nonzero elements in $\mathbf{w}^{(k)}$.

Fig. 7 illustrates the variation process of the metric $R^{(k)}$ on four real-word datasets, from which we can see that, in each iteration of the loop, over 60% of the nonzero elements maintain their positions. We call this feature **relative stability** of nonzero element positions.

Given that the positions of nonzero elements exhibit **relative stability**, this indicates that in the process of adjacent continuous multiple rounds of iterations, only a subset of nonzero elements with fixed positions are being updated. For example, as shown in Fig. 6, in the first round of iteration, only w_i and w_j are predicted as nonzero elements, while all other elements are zero. And in the first k rounds of iteration, only w_i and w_j necessitate updates. This situation renders the prediction of the remaining zero elements somewhat unnecessary. Although the computational complexity of prediction behavior is not high, the more rounds required to update these nonzero elements, the more computing resources are wasted, especially for large-scale networks.

Acceleration Strategy Design: To alleviate the situation above, we proposed to focus on updating the nonzero weights filtered out by $\overline{z}_i^{(k)}$ and $\underline{z}_i^{(k)}$ until convergence initially, as shown in Stage 1 in Fig. 8. The complete organization of FKSC algorithm and its computational complexity analysis are detailed in A.4.

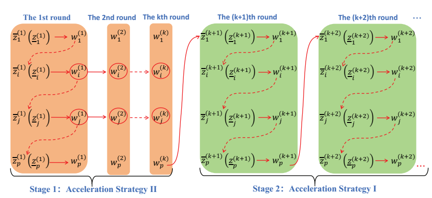


Figure 8: Overview of the two-stage acceleration strategy.

5 EXPERIMENT

Overall, our experimentation encompasses an initial Kernel Sparse Coding Dictionary Learning (KSCDL) training phase and a subsequent Fast Kernel Sparse Coding (FKSC) testing phase. The specific details of the experimental setup, including the dataset, comparison algorithms, and evaluation metrics, can be found in A.5. Within the initial KSCDL training phase, we replace KSC with FKSC, verifying the convergence of the KSCDL method and analyzing the impact of various hyper-parameters on the KSCDL objective value in A.6. Moving on to the subsequent FKSC testing phase, we proceed to validate the effectiveness of the acceleration strategy within FKSC, and compare its performance in the context of data compression (the specific implementation steps are outlined in A.7) with other existing algorithms. The corresponding results are shown in the following.

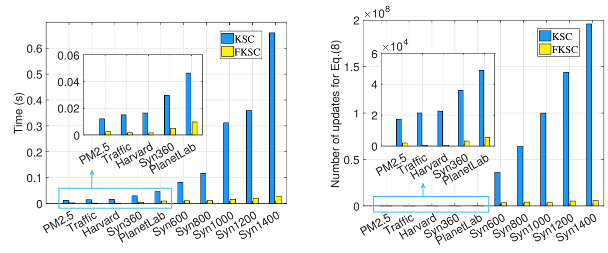
Effectiveness of Acceleration Strategy: The effectiveness of the acceleration strategy is manifested in two aspects: accuracy and processing time. In the following, we verify that the proposed FKSC method can accelerate processing time while ensuring there is no additional loss of accuracy.

Given the limited scale of real datasets, which may hinder the effectiveness of our acceleration strategy, thus, we integrate synthetic data, gradually increasing the data scale, to verify the scalability performance of the proposed algorithm. Specifically, we generate synthetic data, denoted as Syn600, Syn800, Syn1000, Syn1200 and Syn1400 respectively, following the generation method of Syn360 in A.5. The data sizes used for training in them are $600 \times 1200, 800 \times 1600, 1000 \times 2000, 1200 \times 2400, 1400 \times 2800$, and the data sizes for testing are $600 \times 600, 800 \times 800, 1000 \times 1000, 1200 \times 1200, 1400 \times 1400$, with the dictionary matrix configured as a square and hyperparameters set to $\lambda = 0.01$ and $\sigma = 20$ uniformly.

The experimental results are shown in Table 1 and Fig. 9, from which we can observe that

• The two-stage accelerated strategy in FKSC does not exert significantly detrimental effect on accuracy. As shown in Table 1, the objective values remain consistent between FKSC and KSC across the evaluated datasets.

• The FKSC method can significantly reduce processing time. Fig. 9 (a) shows that FKSC has effectively improved the efficiency of KSC, achieving a remarkable reduction in processing time by up to 90%, for each column in sparse weight matrix $\mathbf{W}$. This improvement is attributed to FKSC's strategy of calculating the upper and lower bounds of intermediate variables, thereby substantially circumventing their updates, as illustrated in Fig. 9 (b).



(a) (b)

Figure 9: FKSC's scalability experiments. For each column in sparse weight matrix $\mathbf{W}$: (a) the average processing time; (b) the average number of updates for (8).

• As the data size increases, the acceleration effect of FKSC becomes increasingly evident. As shown in Fig. 9 (a), the processing time for both KSC and FKSC increases in tandem with the expansion of data scale. Notably, the KSC method experiences a swift surge in processing time, contrasted by a more gradual increment for the FKSC method. As a result, the gap in processing time between KSC and FKSC becomes increasingly pronounced as the data scale increases.

Table 1: The objective values for KSC and FKSC

	PM2.5	Traffic	Harvard	Syn360	PlanetLab	Syn600	Syn800	Syn1000	Syn1200	Syn1400
KSC	0.2761	0.1378	0.3877	1.8439	0.8328	1.6057	2.1432	2.6820	3.2092	3.7574
FKSC	0.2761	0.1378	0.3877	1.8439	0.8328	1.6057	2.1432	2.6820	3.2092	3.7574

Compression Performance: Given that FKSC effectively converts dense data into sparse data comprising merely a few non-zero elements, we delve into its application within the realm of data compression. The intricate implementation details are comprehensively presented in A.7.

The Compression Ratio (CR) is determined by FKSC, which hinges critically on the sparse penalty parameter λ. Specifically, as stated in (9), the larger the value of λ, the greater the interval $[-\lambda, \lambda]$ becomes. Consequently, more intermediate variables $z_i (i = 1, 2, \cdots, p)$ fall within this interval, leading to a higher number of zero weights. As a result, $\mathbf{w}$ becomes sparser, and the compression ratio increases. However, there is a trade-off as an excessively large λ can render $\mathbf{w}$ overly sparse, potentially compromising the accuracy of data reconstruction.

Motivated by the experiments detailed in A.6, we take $\sigma = 10, 4, 11, 17$ and 10 for PM2.5, Traffic, Harvard, Syn360 and PlanetLab respectively, and confine the penalty parameter λ within the interval $[0.001, 0.1]$ to ensure that the sparse weight matrix $\mathbf{W}$ is not too sparse to recover the original data $\mathbf{Y}$. In Table 2, we detail the compression performance parameters of various algorithms across five datasets at four distinct compression ratios (CRs), with the optimal and sub-optimal performance parameters highlighted in bold. From which, several key observations emerge:

• Superiority of FKSC over CD: With the exception of the PM2.5 dataset at $\mathrm{CR} = 26.0293$, the FKSC method consistently outperforms CD in all three performance metrics. Notably, in the Traffic data at $\mathrm{CR} = 57.3321$, i.e., the data size is compressed to 1.75% of the original scale, the NMAE, NMAE, NECR metrics of FKSC reach 0.1764, 0.1259, 6057 respectively, achieving a remarkable 21% relative improvement in NRMSE, 47% in NMAE, and an astonishing three times increase in NECR. This underscores the significant enhancement in sparse coding performance brought about by the incorporation of kernel trick.

• Competitive performance of FKSC compared to KSC: It is evident that these restoration performance parameters of FKSC closely align with those of KSC across five datasets at four different CRs. This indicates that the two-stage acceleration strategy in FKSC exhibits virtually no adverse impact on data compression.

Table 2: Compression performance of FKSC and other comparison methods.

Traffic	CR=30.5866			CR=40.3351			CR=50.1481			CR=57.3321		
	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR
CD	0.1902	0.1569	2967	0.2012	0.1708	2285	0.2077	0.1781	2238	0.2135	0.1857	1900
FKSC	**0.1723**	**0.1242**	**6077**	**0.1742**	**0.1250**	5980	**0.1760**	**0.1251**	**6184**	**0.1764**	**0.1259**	**6057**
KSC	**0.1723**	**0.1242**	**6075**	**0.1742**	**0.1250**	**5982**	**0.1760**	**0.1251**	**6183**	**0.1764**	**0.1259**	**6057**
KFMC	0.2026	0.1611	4007	0.2018	0.1596	4240	0.1986	0.1525	4600	0.2092	0.1619	4189
KSR-L_{21}	0.1760	0.1290	5989	0.1857	0.1339	**6233**	0.2048	0.1444	5476	0.2476	0.1761	4402
LPM	0.2799	0.2323	2589	0.2815	0.2342	2430	0.2709	0.2253	2534	0.2797	0.2320	2665
PM2.5	CR=26.0293			CR=32.1060			CR=41.9058			CR=51.1438		
	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR
CD	0.2581	0.2078	**2168**	0.2877	0.2373	1689	0.3075	0.2565	1377	0.3282	0.2769	1303
FKSC	**0.2519**	**0.2060**	**2122**	**0.2575**	**0.2122**	**2025**	**0.2625**	**0.2153**	**2077**	**0.2665**	**0.2195**	**2057**
KSC	**0.2519**	**0.2060**	**2122**	**0.2575**	**0.2122**	**2023**	**0.2625**	**0.2153**	**2078**	**0.2665**	**0.2195**	**2057**
KFMC	0.2643	0.2248	1601	0.2656	0.2285	1567	0.2861	0.2447	1612	0.2711	0.2304	1728
KSR-L_{21}	0.2803	0.2358	1576	0.4054	0.3152	1249	0.3110	0.2634	1523	0.3539	0.2867	1426
LPM	0.3781	0.3421	910	0.3471	0.3127	1044	0.3308	0.2923	1255	0.3135	0.2759	1352
Harvard	CR=40.3763			CR=54.1633			CR=63.2912			CR=74.4482		
	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR
CD	0.2083	0.1774	3924	0.2181	0.1915	3187	0.2365	0.2173	2884	0.2435	0.2261	2737
FKSC	**0.2028**	**0.1654**	**4971**	**0.2057**	**0.1675**	**4481**	**0.2106**	**0.1760**	**3868**	**0.2130**	**0.1786**	**3836**
KSC	**0.2028**	**0.1654**	**4971**	**0.2057**	**0.1675**	**4481**	**0.2106**	**0.1760**	**3868**	**0.2130**	**0.1786**	**3836**
KFMC	0.3119	0.2163	2907	0.2686	0.2282	3298	0.3536	0.3017	3137	0.4250	0.3129	3215
KSR-L_{21}	0.2726	0.2232	2999	0.3028	0.2355	2760	0.3265	0.3046	2599	0.3938	0.2868	2265
LPM	0.3649	0.3552	1950	0.3321	0.3089	2450	0.3334	0.3140	2324	0.3138	0.2892	2541
Syn360	CR=54.1557			CR=81.2641			CR=105.4173			CR=140.4878		
	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR
CD	0.1086	0.0880	20548	0.1140	0.0927	19136	0.1195	0.0975	17838	0.1253	0.1031	16500
FKSC	**0.1024**	**0.0824**	**22255**	**0.1036**	**0.0834**	**22044**	**0.1043**	**0.0839**	**22014**	**0.1048**	**0.0843**	**21711**
KSC	**0.1024**	**0.0824**	**22255**	**0.1036**	**0.0834**	**22044**	**0.1043**	**0.0839**	**22014**	**0.1048**	**0.0843**	**21711**
KFMC	0.1401	0.1130	16127	0.1539	0.1232	14794	0.1541	0.1244	14642	0.1398	0.1134	15768
KSR-L_{21}	0.1121	0.0905	19931	0.1171	0.0941	19327	0.1076	0.0856	21947	0.1093	0.0874	21342
LPM	0.2460	0.2030	8578	0.2528	0.2098	8082	0.2557	0.2127	8109	0.2571	0.2161	7515
PlanetLab	CR=49.2513			CR=60.4177			CR=68.2878			CR=81.5557		
	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR	NRMSE	NMAE	NECR
CD	0.2360	0.1597	31726	0.3221	0.2411	19466	0.2599	0.1881	23786	0.3360	0.2543	18120
FKSC	**0.2236**	**0.1469**	**36608**	**0.2308**	**0.1575**	32540	**0.2216**	**0.1472**	**35106**	**0.2281**	**0.1560**	32599
KSC	**0.2236**	**0.1469**	**36606**	**0.2308**	**0.1575**	32540	**0.2216**	**0.1472**	**35098**	**0.2281**	**0.1560**	32597
KFMC	0.2369	0.1666	32553	0.2311	**0.1557**	**36053**	0.2417	0.1670	33887	0.2380	0.1633	**34773**
KSR-L_{21}	0.2397	0.1633	35040	0.2365	0.1600	**35159**	0.2768	0.1869	33715	0.2456	0.1643	**34437**
LPM	0.3563	0.2949	19655	0.3083	0.2402	24854	0.3147	0.2499	23261	0.2999	0.2318	25542

• Advantage of FKSC over other kernel sparse coding methods: In the totality of 20 experimental setups, FKSC fails to achieve the top performance in only 3 instances, signifying its overwhelming dominance in most scenarios. For instance, in the Harvard data at $\mathrm{CR} = 54.1633$, i.e., the data size is compressed to 1.85% of the original scale, the NMAE, NMAE, NECR metrics of FKSC reach 0.2057, 0.1675, 4481 respectively. When KSC is excluded from consideration, KFMC emerges as the sub-optimal method. Compared to KFMC, FKSC exhibits a substantial 30% relative improvement in NRMSE, 36% in NMAE, and 1.36 times boost in NECR. This may be attributed to the solution of FKSC, which boasts a convergence guarantee and facilitates the optimization process in achieving an exceptional stationary point.

6 Conclusion

In this paper, we have addressed the limitations of traditional sparse coding algorithms when dealing with nonlinear real-world signals. By leveraging RBF kernel to implicitly increase the dimensionality of the original data to enhance its separability, our proposed kernel sparse coding method enables more effective sparse representations and provides a solution with convergence guarantees based on the principle of coordinate descent. To further optimize the computational efficiency, we introduced a novel two-stage acceleration strategy. The strategy is theoretically underpinned by the insight that updates to zero weights can be skipped and empirically supported by the observation that the positions of nonzero weights are relatively stable. This innovation allows the optimization process to be significantly accelerated. Experimental results validate the effectiveness of our approach. The two-stage acceleration strategy demonstrates a remarkable reduction in processing time by up to 90%. Additionally, our method shows superior performance compared to both traditional linear sparse coding methods and other kernel sparse coding techniques, with significantly lower values of NMAE when CR is high.

REFERENCES

Chenglong Bao, Hui Ji, Yuhui Quan, and Zuowei Shen. L0 norm based dictionary learning by proximal methods with global convergence. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 3858–3865, 2014.

Samy Bengio, Fernando Pereira, Yoram Singer, and Dennis Strelow. Group sparse coding. *Advances in neural information processing systems*, 22, 2009.

Matthew Chalk, Olivier Marre, and Gašper Tkačik. Toward a unified theory of efficient, predictive, and sparse coding. *Proceedings of the National Academy of Sciences*, 115(1):186–191, 2018.

Xinyu Chen, Yixian Chen, and Zhaocheng He. Urban traffic speed dataset of Guangzhou, China. 2018. URL `https://zenodo.org/records/1205229`.

Jicong Fan and Madeleine Udell. Online high rank matrix completion. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 8690–8698, 2019.

Jicong Fan, Chengrun Yang, and Madeleine Udell. Robust non-linear matrix factorization for dictionary learning, denoising, and clustering. *IEEE Trans. Signal Process.*, 69:1755–1770, 2021.

Yasuhiro Fujiwara, Yasutoshi Ida, Hiroaki Shiokawa, and Sotetsu Iwamura. Fast lasso algorithm via selective coordinate descent. In *Conference on Artificial Intelligence, AAAI*, pp. 1561–1567, 2016.

Shenghua Gao, Ivor Wai-Hung Tsang, and Liang-Tien Chia. Kernel sparse representation for image classification and face recognition. In *European Conference on Computer Vision, ECCV*, volume 6314, pp. 1–14, 2010.

Shenghua Gao, Ivor Wai-Hung Tsang, and Liang-Tien Chia. Sparse representation with kernels. *IEEE Trans. Image Process.*, 22(2):423–434, 2013.

Changyong Guo, Zhaoxin Zhang, Jinjiang Li, Xuesong Jiang, Jun Zhang, and Lei Zhang. Robust visual tracking using kernel sparse coding on multiple covariance descriptors. *ACM Trans. Multim. Comput. Commun. Appl.*, 16(1s):20:1–20:22, 2020.

Babak Hosseini, Felix Hülsmann, Mario Botsch, and Barbara Hammer. Non-negative kernel sparse coding for the analysis of motion data. In *Artificial Neural Networks and Machine Learning, ICANN*, volume 9887, pp. 506–514, 2016.

Minyoung Kim. Efficient kernel sparse coding via first-order smooth optimization. *IEEE Trans. Neural Networks Learn. Syst.*, 25(8):1447–1459, 2014.

Jonathan Ledlie, Paul Gardner, and Margo I. Seltzer. Network coordinates in the wild. In *Networked Systems Design and Implementation NSDI*, 2007.

Honglak Lee, Alexis J. Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems, NeurIPS*, pp. 801–808, 2006.

Wenjie Li, Francesca Bassi, and Michel Kieffer. Sparse random linear network coding for data compression in wsns. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 2729–2733, 2016.

Yuan Liu, Stéphane Canu, Paul Honeine, and Su Ruan. Mixed integer programming for sparse coding: Application to image denoising. *IEEE Trans. Computational Imaging*, 5(3):354–365, 2019.

Zhiwu Lu, Xin Gao, Liwei Wang, Ji-Rong Wen, and Songfang Huang. Noise-robust semi-supervised learning by large-scale sparse coding. In Blai Bonet and Sven Koenig (eds.), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pp. 2828–2834. AAAI Press, 2015.

Hien Van Nguyen, Vishal M. Patel, Nasser M. Nasrabadi, and Rama Chellappa. Design of non-linear kernel dictionaries for object recognition. *IEEE Trans. Image Process.*, 22(12):5123–5135, 2013.

Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pp. 40–44, 1993.

Daniel Pimentel-Alarcón, Gregory Ongie, Laura Balzano, Rebecca Willett, and Robert Nowak. Low algebraic dimension matrix completion. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 790–797. IEEE, 2017.

Yuqing Qian, Yijie Ding, Quan Zou, and Fei Guo. Multi-view kernel sparse representation for identification of membrane protein types. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 20(2): 1234–1245, 2023.

Yuhui Quan, Chenglong Bao, and Hui Ji. Equiangular kernel dictionary learning with applications to dynamic texture analysis. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 308–316, 2016.

Tenkasi V Ramabadran and Deepen Sinha. Speech data compression through sparse coding of innovations. *IEEE Transactions on Speech and Audio Processing*, 2(2):274–284, 1994.

Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks, ICANN*, volume 1327, pp. 583–588, 1997.

Henry Schütze, Erhardt Barth, and Thomas Martinetz. Learning efficient data representations with orthogonal sparse coding. *IEEE Trans. Computational Imaging*, 2(3):177–189, 2016. doi: 10.1109/TCI.2016.2557065. URL https://doi.org/10.1109/TCI.2016.2557065.

Henry Schütze, Erhardt Barth, and Thomas Martinetz. Learning efficient data representations with orthogonal sparse coding. *IEEE transactions on computational imaging*, 2(3):177–189, 2016.

Guolong Shi, Yigang He, and Chaolong Zhang. Feature extraction and classification of cataluminescence images based on sparse coding convolutional neural networks. *IEEE Trans. Instrum. Meas.*, 70:1–11, 2021. doi: 10.1109/TIM.2020.3023508. URL https://doi.org/10.1109/tim.2020.3023508.

J Michael Steele. *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities*. Cambridge University Press, 2004.

Jijun Tong, Yingjie Zhao, Peng Zhang, Lingyu Chen, and Lurong Jiang. MRI brain tumor segmentation based on texture features and kernel sparse coding. *Biomed. Signal Process. Control.*, 47: 387–392, 2019. doi: 10.1016/J.BSPC.2018.06.001. URL https://doi.org/10.1016/j.bspc.2018.06.001.

P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.

Yaqing Wang, James T. Kwok, and Lionel M. Ni. Generalized convolutional sparse coding with unknown noise. *IEEE Trans. Image Process.*, 29:5386–5395, 2020.

Yi Wang, Qixin Chen, Chongqing Kang, Qing Xia, and Min Luo. Sparse and redundant representation-based smart meter data compression and pattern extraction. *IEEE Transactions on Power Systems*, 32(3):2142–2151, 2016.

Yijing Watkins, Oleksandr Iaroshenko, Mohammad Sayeh, and Garrett Kenyon. Image compression: Sparse coding vs. bottleneck autoencoders. In *2018 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, pp. 17–20. IEEE, 2018.

Jianchao Yang, Kai Yu, Yihong Gong, and Thomas S. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Society Conference on Computer Vision and Pattern Recognition CVPR*, pp. 1794–1801, 2009.

Lu Yang, Hong Cheng, Jianan Su, and Xuelong Li. Pixel-to-model distance for robust background reconstruction. *IEEE Trans. Circuits Syst. Video Technol.*, 26(5):903–916, 2016.

Taku Yutani, Oak Yono, Tatsu Kuwatani, Daisuke Matsuoka, Junji Kaneko, Mitsuko Hidaka, Takafumi Kasaya, Yukari Kido, Yoichi Ishikawa, Toshiaki Ueki, and Eiichi Kikawa. Super-resolution and feature extraction for ocean bathymetric maps using sparse coding. *Sensors*, 22(9):3198, 2022.

Yu Zheng, Xiuwen Yi, Ming Li, Ruiyuan Li, Zhangqing Shan, Eric Chang, and Tianrui Li. Forecasting fine-grained air quality based on big data. In *International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pp. 2267–2276. ACM, 2015. URL `https://doi.org/10.1145/2783258.2788573`.

Rui Zhu, Bang Liu, Di Niu, Zongpeng Li, and Hong Vicky Zhao. Network latency estimation for personal devices: A matrix completion approach. *IEEE/ACM Trans. Netw.*, 25(2):724–737, 2017.

A Appendix

A.1 Related work

Sparse coding serves as a pivotal branch within deep learning (Yang et al., 2009), endeavoring to discover a sparse representation of input data in the form of a linear combination of basic atoms. This pursuit achieves multiple objectives: data compression, enhancement of computational efficiency, and uncovering salient features of the data, thereby finding widespread applications across diverse domains.

However, recent studies (Pimentel-Alarcón et al., 2017; Fan et al., 2021; Fan & Udell, 2019) have indicated that many real-world signals do not possess an inherent linear structure, hence traditional linear sparse coding methods may fail to be effective (Yang et al., 2016). Consequently, a few scholars have proposed the use of kernel tricks to implicitly elevate the dimensionality of the input data, thereby enhancing the applicability of sparse coding methods to complex data. Kernel sparse coding is an extension of linear sparse coding. Existing research on kernel sparse coding problems can be roughly divided into two categories based on the form of constraint on the sparse matrix in their mathematical models: the l_0 norm-based approach and the l_1 norm-based approach.

The kernel sparse coding problem formulated with the l_0 norm is inherently NP-hard and can only be solved by using heuristic algorithms. For instance, the Kernel Orthogonal Matching Pursuit (KOMP) (Nguyen et al., 2013) algorithm iteratively selects dictionary atoms that exhibit the highest correlation with the current residual during the sparse optimization process until a predetermined sparsity level is achieved. Alternatively, the Linearized Proximal Method (LPM) (Quan et al., 2016) leverages block coordinate descent, initially obtaining a closed-form solution through the proximal gradient method, followed by a brute-force selection of the weights with the largest absolute values to satisfy the sparsity constraint. These heuristic algorithms are collectively noted for their rapid sparse optimization capabilities. However, **they fall short by lacking convergence guarantees**, which means they cannot assure the attainment of the optimal solution and are susceptible to getting trapped in local optima.

In contrast, the l_1 norm offers a convex alternative that is easier to handle mathematically but still presents challenges due to its non-differentiability. The Kernel Feature-sign Search (KFSS) (Gao et al., 2013) algorithm addresses this issue by taking an active approach to guessing the signs of the coefficient weights. This strategic method is based on the observation that the sign of a coefficient is closely related to the correlation between the dictionary atom and the residual. By correctly guessing the signs, KFSS can simplify the optimization problem and work towards finding an analytical solution that satisfies the sparsity requirement. The First-Order Smooth Optimization (FOSO) (Kim, 2014) algorithm tackles the non-differentiability of the l_1 norm by approximating it with smooth functions. This approximation allows the use of conventional gradient-based optimization techniques, which are otherwise inapplicable to non-differentiable problems. The smooth approximation serves as a surrogate that enables the derivation of an analytical solution while still promoting sparsity in the solution. Both KFSS and FOSO contribute to the broader field of kernel sparse coding by providing methods that can yield sparse solutions with convergence guarantees. However, **they face challenges in terms of the scalability of optimization complexity and computational speed**. Therefore, this paper aims to explore a new fast kernel sparse coding method with convergence guarantees.

A.2 KSCDL Algorithm

Algorithm 1 KSCDL

Input: $\mathbf{Y}$, κ, λ, t_{max}
1: **Initialize**: $\mathbf{X} \sim \mathcal{N}(0,1)$, $t = 0$
2: **repeat**
3: $t = t + 1$
4: Update sparse coding matrix $\mathbf{W}^{(k)}$ through (9)
5: Update dictionary matrix $\mathbf{X}^{(k)}$ through (11)
6: **until** the convergence condition is satisfied or $t = t_{max}$
Output: $\mathbf{W}$, $\mathbf{X}$

The convergence condition in Algorithm 1 refers that the relative error $RE = \|\mathbf{X}^{(k)} - \mathbf{X}^{(k-1)}\|_F / \|\mathbf{X}^{(k-1)}\|_F < \varepsilon$ for any $\varepsilon > 0$. The following Theorem shows this algorithm converges.

Theorem: Algorithm 1 converges to a stationary point.

Proof. Denote the objective function in (3) as $\mathcal{L}(\mathbf{X}, \mathbf{W})$. On the one hand, when $\mathbf{W}$ is fixed, $\mathcal{L}(\mathbf{X}, \mathbf{W})$ is twice differentiable with respect to $\mathbf{X}$, and the gradient $G_{\mathbf{X}}$ is Lipschitz continuous with Lipschitz constants $L_{\mathbf{X}}$, then the Taylor expansion of $\mathcal{L}(\mathbf{X}, \mathbf{W})$ around $\mathbf{X}^{(k-1)}$ is

$$\begin{aligned} \mathcal{L}(\mathbf{X}^{(k)}, \mathbf{W}) = & \mathcal{L}(\mathbf{X}^{(k-1)}, \mathbf{W}) + \langle G_{\mathbf{X}}, \mathbf{X}^{(k)} - \mathbf{X}^{(k-1)} \rangle \\ & + \frac{1}{2} \mathrm{vec}(\mathbf{X}^{(k)} - \mathbf{X}^{(k-1)})^T H_{\mathbf{X}} \mathrm{vec}(\mathbf{X}^{(k)} - \mathbf{X}^{(k-1)}), \end{aligned} \tag{17}$$

where $H_{\mathbf{X}}$ is the Hessian matrix of $\mathcal{L}(\mathbf{X}, \mathbf{W})$ with respect to $\mathbf{X}$. $\langle \cdot, \cdot \rangle$ is the trace of the inner product of matrices, i.e., $\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{trace}(\mathbf{A}^T \mathbf{B})$. Substituting (11) into (17) yields

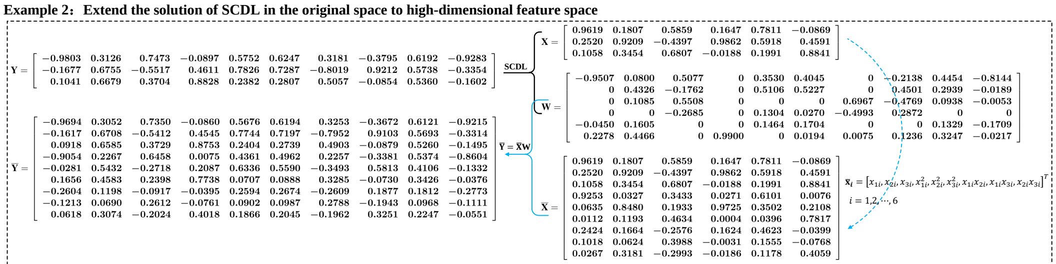
$$\begin{aligned} \mathcal{L}(\mathbf{X}^{(k)}, \mathbf{W}) \leq & \mathcal{L}(\mathbf{X}^{(k-1)}, \mathbf{W}) + \langle G_{\mathbf{X}}, -\mu G_{\mathbf{X}} \rangle + \frac{\lambda_{\max}(H_{\mathbf{X}})}{2} \mathrm{vec}(-\mu G_{\mathbf{X}})^T \mathrm{vec}(-\mu G_{\mathbf{X}}) \\ \leq & \mathcal{L}(\mathbf{X}^{(k-1)}, \mathbf{W}) - \frac{2\mu - \mu^2 L_{\mathbf{X}}}{2} \|G_{\mathbf{X}}\|_F^2, \end{aligned} \tag{18}$$

where $\lambda_{\max}(H_{\mathbf{X}})$ is the maximum eigenvalue of Hessian matrix $H_{\mathbf{X}}$. Clearly, $\frac{2\mu - \mu^2 L_{\mathbf{X}}}{2} \geq 0$ if $0 \leq \mu \leq \frac{2}{L_{\mathbf{X}}}$. That is to say, when $\mathbf{X}$ is updated according to equation (11), the objective function $\mathcal{L}(\mathbf{X}, \mathbf{W})$ is decreasing and bounded below by 0. Hence, the objective function $\mathcal{L}(\mathbf{X}, \mathbf{W})$ with respect to $\mathbf{X}$ is convergent.

On the other hand, when $\mathbf{X}$ is fixed, the kernel sparse coding subproblem (4) constitutes a Lasso problem, with the explicit solution (9) obtained directly through the coordinate descent algorithm. Thus, similar to Fujiwara et al. (2016) and following the convergence analysis presented in Tseng (2001), the objective function $\mathcal{L}(\mathbf{X}, \mathbf{W})$ with respect to $\mathbf{W}$ is convergent. $\square$

A.3 Discussion on extending the solution of SCDL to higher-dimensional feature space

Example 2: Extend the solution of SCDL in the original space to high-dimensional feature space



In Example 1, the SCDL result of input data $\mathbf{Y}$ yields the dictionary matrix $\mathbf{X}$ and the sparse matrix $\mathbf{W}$. In addition, $\mathbf{Y}$ is vertically expanded to high-dimensional data $\widetilde{\mathbf{Y}}$, and the SCDL result of $\widetilde{\mathbf{Y}}$ yields the dictionary $\widetilde{\mathbf{X}}$ and the sparse matrix $\widetilde{\mathbf{W}}$. In this case, the approximation error $AE_{high1} = \frac{1}{2}\|\widetilde{\mathbf{Y}} - \widetilde{\mathbf{X}}\widetilde{\mathbf{W}}\|_F^2 = 0.3625$. In Example 2, suppose we vertically expand the dictionary $\mathbf{X}$ (obtained from Example 1) to dictionary $\overline{\mathbf{X}}$ in the high-dimensional feature space, we can derive the high-dimensional approximate data as $\overline{\mathbf{Y}} = \overline{\mathbf{X}}\mathbf{W}$, where $\mathbf{W}$ is obtained from Example 1. In this case, the approximation error in the high-dimensional feature space is $AE_{high2} = \frac{1}{2}\|\widetilde{\mathbf{Y}} - \overline{\mathbf{Y}}\|_F^2 = 4.8904$, which is considerably higher than AE_{high1}. That is to say, when directly extending the results of SCDL in the original space to a high-dimensional feature space, the data approximation effect is not as good as performing SCDL directly in the high-dimensional feature space. In other words, the solutions of (3) should not be obtained by simply extending the solutions of (1). Consequently, it is necessary to explore novel approaches to solve the KSCDL problem (3).

A.4 FKSC ALGORITHM

The FKSC method with a two-stage acceleration strategy is described in Algorithm 2. The core strategies of FKSC includes: Stage 1: Prioritizing updates of initially predicted nonzero elements (lines 3-11); Stage 2: Safely skipping updates for zero elements in **w** (lines 14-16).

Computational complexity analysis. Assuming the FKSC method requires T_1 rounds of iterations in Stage 1 and T_2 rounds of iterations in Stage 2, while the KSC method requires T rounds of iterations for the weight vector to converge.

In Stage 1 of FKSC, assuming that in the 1st round of iteration, there are k_1 elements are predicted to be nonzero and require T_1 iterations to converge. Thus, the total computational complexity of Stage 1 is $O(p + k_1 p T_1)$, where $O(p)$ represents the total computational complexity of calculating the bounds in the 1st round of iteration, and $O(k_1 p T_1)$ represents the total computational complexity of the k_1 predicted nonzero elements over T_1 rounds of iteration.

In Stage 2 of FKSC, assuming that the entire weight vector requires T_2 iterations to converge, and on average, k_0 elements are predicted to be zero in per round of iteration. Therefore, the total computational complexity of Stage 2 is $O(pT_2(p - k_0 + 1))$, where $O(pT_2)$ is the total computational complexity of calculating the bounds throughout the process, and $O((p - k_0)pT_2)$ is the total computational complexity of the remaining $(p - k_0)$ elements, excluding the predicted k_0 zero elements, to undergo T_2 rounds of iterations.

Hence, the total computational complexity of the entire FKSC algorithm is $O(p(1 + k_1T_1 + T_2(p - k_0 + 1)))$. If no acceleration strategies are adopted and the standard coordinate descent algorithm is used for updating iterations directly, then the overall computational complexity would be $O(p^2(T_1 + T_2))$. Generally speaking, T is not much different from $T_1 + T_2$, then the two-stage acceleration strategy has the potential to speed up the processing time of kernel sparse coding.

Algorithm 2 Complete organization of FKSC

Input: $\mathbf{X}$, $\mathbf{y}$, κ,
1: Compute $\kappa(\mathbf{y}, \mathbf{X})$, $\kappa(\mathbf{X}, \mathbf{X})$
2: **Initialize**: $\mathbf{w}^{(0)} = \mathbf{0}$, $\mathbf{z}^{(0)} = \mathbf{0}$, $\Omega = \{1, \cdots, p\}$, $\widetilde{\Omega} = \emptyset$
3: **for** each $i \in \Omega$ **do**
4:     Compute $\overline{z}_i^{(k)}$ and $\underline{z}_i^{(k)}$ by (14)
5: **end for**
6: **if** $\underline{z}_i^{(k)} > \lambda$ or $\overline{z}_i^{(k)} < -\lambda$ **then**
7:     $\widetilde{\Omega} = \widetilde{\Omega} \cup \{i\}$
8: **end if**
9: **repeat**
10:     Update $w_i^{(k)}$ through (9) for each $i \in \widetilde{\Omega}$
11: **until w** converges
12: **repeat**
13:     **for** each $i \in \Omega$ **do**
14:         Compute $\overline{z}_i^{(k)}$ and $\underline{z}_i^{(k)}$ by (14)
15:         **if** $-\lambda \leq \underline{z}_i^{(k)} \leq \overline{z}_i^{(k)} \leq \lambda$ **then**
16:             $w_i^{(k)} = 0$
17:         **else**
18:             Update $w_i^{(k)}$ through (9)
19:         **end if**
20:     **end for**
21: **until w** converges
Output: **w**

A.5 EXPERIMENTAL SETUPS

A.5.1 DATASETS

The datasets mainly utilized encompass four real-world datasets and one synthetic dataset. Acknowledging the dimensional constraints of the real-world datasets, we specifically introduce a synthetic dataset of an tailored dimension (referred to as "Syn360"), to ensure a progressive scaling of dimensions across the datasets, thereby facilitate our comprehensive observation and analysis of the experimental outcomes.

• **PM2.5** (Zheng et al., 2015) records the air quality data collected by Microsoft Research's Urban Computing team for a year (from May 1, 2014 to April 30, 2015) in the Urban Air project. The dataset covers four major cities in China (Beijing, Tianjin, Guangzhou, and Shenzhen) and 39 adjacent cities within a 300-kilometer radius.

• **Traffic** (Chen et al., 2018) encompasses traffic speed observations from 214 anonymous road segments, primarily comprising urban highways and main thoroughfares, spanning a two-month period from August 1, 2016, to September 30, 2016. The data is recorded every 10 minutes, originating from Guangzhou, China.

• **Harvard** (Ledlie et al., 2007) contains the data of application-level RTT, gathered from interactions among 226 Azureus clients over a span of 72 hours.

• **PlanetLab** (Zhu et al., 2017) consists of RTT measurements between 490 nodes in the PlanetLab network across 18 time slices.

• **Syn360** (similar to (Fan et al., 2021)) is a synthetic data with each column generated by $\mathbf{y} = \psi(\mathbf{s}) = \mathbf{P}\widetilde{\mathbf{s}}$, where $\mathbf{s} = [s_1, s_2, s_3]^T \sim \mathcal{U}(-1, 1)$; $\psi \in \{\psi^1, \cdots, \psi^{15}\}$ is an order-3 polynomial mapping with $\mathbf{P} \in \{\mathbf{P}^1, \cdots, \mathbf{P}^{15}\} \subset R^{360\times 20} \sim \mathcal{N}(0, 1)$. $\widetilde{\mathbf{s}} = [1, s_1, s_2, s_3, s_1^2, s_1s_2, s_1s_3, s_2^2, s_2s_3, s_3^2, s_1^3, s_1^2s_2, s_1^2s_3, s_1s_2s_3, s_2^3, s_2^2s_1, s_2^2s_3, s_3^3, s_3^2s_1, s_3^2s_2]^T \in R^{20}$; For each $\mathbf{P}^i, i = 1, \cdots, 18$, we randomly generate 100 $\mathbf{y}$s.

In Table 3, we present the size of the data selected from each dataset, used for training the dictionary, and for testing the performance of the proposed FKSC algorithm.

Table 3: The processed data size for training and testing

	PM2.5	Traffic	Harvard	Syn360	PlanetLab
Training	174×920	214×1440	226×1130	360×1000	490×1470
Testing	174×184	214×288	226×226	360×500	490×490

A.5.2 Competitors

The comparison algorithms for KSC and FKSC include

• **CD** (Fujiwara et al., 2016): the coordinate descent algorithm for traditional linear sparse coding problem, based on l_1 regularization in the original input space.

• **KFMC** (Fan & Udell, 2019): the explicit solution approach for kernelized matrix factorization problems, with the weight matrix regularized by the Frobenius norm.

This method results in a weight matrix that is not strictly sparse, but contains many weight factors that are very small. In this paper, for each column of the weight matrix, we preserve weight factors with significant absolute values while setting the others to zero to meet a predetermined sparsity level, thereby facilitating a comparison with our proposed method.

• **KSR-$L_{2,1}$** (Qian et al., 2023): the explicit solution approach for kernel sparse representation, with the weight matrix being constrained via $L_{2,1}$ matrix norm.

Ideally, this method yields a weight matrix that is row-sparse, but the overall sparsity is not high enough, or in other words, there are not enough zero elements. Therefore, to enable a comparative analysis with our proposed method, for each column of the weight matrix, we retain only those weight factors with substantial absolute values in accordance with a preset sparsity level, while nullifying the others to zero.

• **LPM** (Quan et al., 2016): the linearized proximal method, employed for addressing kernel sparse coding problems with l_0 regularization, initiates by leveraging the proximal gradient approach to secure a closed-form solution. Subsequently, it selects and retains the weight factors with the most significant absolute values, aligning with a predefined sparsity level.

A.5.3 METRICS

The compression performance of an algorithm is primarily reflected in two aspects: compression ratio and accuracy. Therefore, we present the following metrics:

• Compression Ratio (CR):

$$\text{CR} = \frac{\text{Size of original data}}{\text{Size of compressed data}}.$$

• Normalized Root Mean Square Error (NRMSE):

$$\text{NRMSE} = \|\mathbf{Y} - \widetilde{\mathbf{Y}}\|_F \Big/ \|\mathbf{Y}\|_F,$$

where $\widetilde{\mathbf{Y}}$ refers to the approximate data obtained by solving the optimization problem (19) for kernel-based algorithms include LPM, KSR-$L_{2,1}$, KFMC, as well as the proposed KSC and FKSC, and $\widetilde{Y} = \mathbf{XW}$ for sparse coding algorithms in the original space like CD.

• Normalized Mean Absolute Error (NMAE):

$$\text{NMAE} = \sum_{(i,j)} |y_{ij} - \widetilde{y}_{ij}| \Big/ \sum_{(i,j)} |y_{ij}|,$$

where $\widetilde{y}_{ij}$ is the element located at the ith row and jth column of matrix $\widetilde{\mathbf{Y}}$.

• Number of Elements Correctly Reconstructed (NECR):

$$\text{NECR} = \sum_{(i,j)} y_{ij}^{\varrho}, \quad y_{ij}^{\varrho} = \begin{cases} 1, & |y_{ij} - \widetilde{y}_{ij}| < \varrho; \\ 0, & \text{Otherwise}, \end{cases}$$

where $\widetilde{y}_{ij}$ is the element located at the ith row and jth column of matrix $\widetilde{\mathbf{Y}}$ and $\varrho = 10^{-3}$.

In these metrics, higher values of CR and NECR indicate superior compression performance by the algorithm. Conversely, lower values of NRMSE and NMAE signify a more effective compression capability.

A.6 EXPERIMENTAL RESULTS

For all experiments, the maximum number of iterations is set to 100, and the algorithm termination tolerance is set to $1e-3$. For KSCDL, the termination tolerance refers to the relative error $RE_{\mathbf{X}} = \|\mathbf{X}^{(k)} - \mathbf{X}^{(k-1)}\|_F / \|\mathbf{X}^{(k-1)}\|_F$, while for FKSC, the termination tolerance refers to the relative error $RE_{\mathbf{W}} = \|\mathbf{W}^{(k)} - \mathbf{W}^{(k-1)}\|_F / \|\mathbf{W}^{(k-1)}\|_F$.

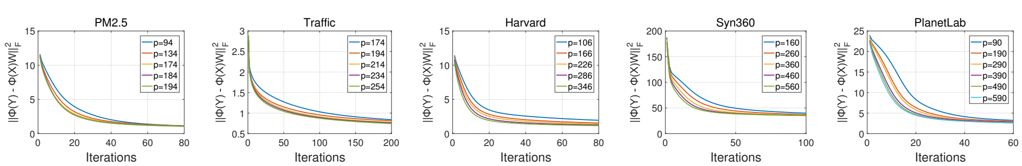


Figure 10: The objective value for different numbers of atoms (i.e., p), with $\lambda = 0.001$ and $\sigma = 2$.

Convergence Verification: To validate the convergence of the KSCDL method outlined in Algorithm 1, we set $\lambda = 0.001$ and $\sigma = 2$. As shown in Fig. 10, the objective values $\|\phi(\mathbf{Y}) - \phi(\mathbf{X})\mathbf{W}\|_F^2$ achieve convergence across the evaluated datasets, even as the number of atoms varies. It is noteworthy that the objective value for Syn360 is comparatively higher than those of the other datasets. This discrepancy arises from the less optimal parameter setting of $\sigma = 2$ for this dataset. Further experimentation demonstrate that adopting a value of σ exceeding 10 leads to significantly enhanced performance.

In addition, we can also observe that as the number of atoms increases, the convergence speed of the objective function gradually accelerates. Nonetheless, once the number of atoms surpasses the number of rows ($174, 214, 226, 360$ and 490 for PM2.5, Traffic, Harvard, Syn360 and PlanetLab, respectively) in the dictionary matrix, the convergence behavior of the objective value is no longer sensitive to the number of atoms. Therefore, setting the number of atoms in the dictionary equal to the number of its rows, that is, configuring the dictionary as a square matrix, is reasonable.

Hyperparameter Analysis: In addition to the number of the dictionary atoms, there are also two hyperparameters that can influence the outcomes of the KSCDL algorithm: the sparsity penalty parameter λ and the RBF kernel hyperparameter σ. With all dictionaries configured as square matrices, Fig. 11 indicates that when λ is fixed and σ is sufficiently large, the objective value remains insensitive to variations in σ. Conversely, when σ is kept constant, the objective function exhibits a tendency of exponential growth as λ increases. To ensure that the objective value of KSCDL remains within a desirable range, we will restrict the value of λ within the interval of $[0.001, 0.1]$, from this point onward.

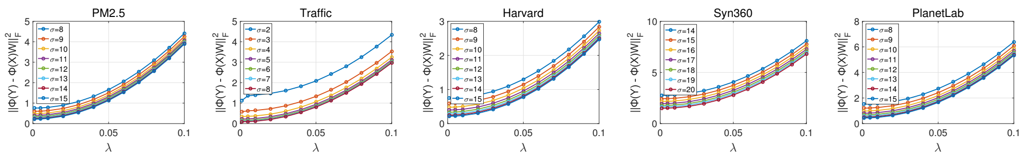


Figure 11: The influence of different λ and σ on objective value.

A.7 Application of FKSC

FKSC transforms dense input data into a sparse representation, thereby enabling solutions for challenges such as data compression, improving of computational efficiency, feature analysis and denoising. Taking data compression as an example, Fig. 12 delineates the application procedure of FKSC. Initially, a dictionary matrix $\mathbf{X}$, which can reflect the underlying structural features of the data, is derived from historical data using the KSCDL algorithm outlined in Algorithm 1, with KSC being replaced by FKSC. In the subsequent stage, the original data undergoes a three-step compression process (i.e., Step1-3) utilizing this dictionary matrix $\mathbf{X}$. Conversely, the compressed data can be approximately reconstructed into its original form through a three-step decompression procedure (i.e., Step4-6). The specific implementation details of each step are outlined below:

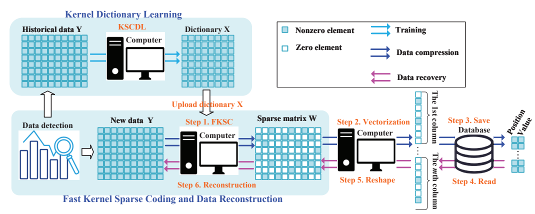


Figure 12: Example for FKSC's application.

- Step 1: Encode new data $\mathbf{Y}$ into sparse matrix $\mathbf{W}$ via the propose FKSC method, leveraging the dictionary matrix $\mathbf{X}$ obtained from the initial stage.
- Step 2: Vectorize the sparse matrix $\mathbf{W}$ to facilitate its subsequent compression.
- Step 3: Save the vectorized $\mathbf{W}$ in a condensed two-column matrix. Column one lists indices of non-zero elements, and column two lists their values.
- Step 4: Decode back the small matrix from the storage unit into a sparse column vector.
- Step 5: Fold the sparse column vector back into the sparse matrix $\mathbf{W}$.
- Step 6: Solve the following optimization problem to obtain the approximate data $\widetilde{\mathbf{Y}}$:

$$\widetilde{\mathbf{Y}} = \arg\min_{\widetilde{\mathbf{Y}}} \frac{1}{2}\|\phi(\widetilde{\mathbf{Y}}) - \phi(\mathbf{X})\mathbf{W}\|_F^2. \tag{19}$$