Toulouse School of Economics

Master 2 Data Science for Social Sciences

Sparse Principal Component Analysis

Session Number: **D3S Session 1**
Project Letter: **B**

Authors:
DRUILHE Théo, PIZZETTA Nathan, SAUE Sigurd

December 2024

Contents

1 Introduction

Dimensionality reduction is essential for analyzing high-dimensional datasets. This project applies Principal Component Analysis (PCA) and Sparse Principal Component Analysis (SPCA) to the **Labeled Faces in the Wild (LFW)** dataset, a benchmark for facial image analysis, to reduce dimensionality while retaining features critical for classification tasks.

The **LFW** dataset comprises grayscale images of seven individuals. A subset of 1,288 images was resized to 50×37 pixels and flattened into 1,850-dimensional feature vectors. The dataset's high dimensionality and seven-class structure make it ideal for evaluating PCA and SPCA.

This project compares the performance of PCA and SPCA in extracting meaningful features from facial data. While PCA captures maximum variance, its dense components lack interpretability. SPCA introduces sparsity, identifying localized features such as eyes and mouth. Structured Sparse PCA (SSPCA) further improves interpretability by incorporating domain-specific structures.

To evaluate these techniques, we analyze their trade-offs in accuracy, interpretability, and efficiency, and use a Support Vector Machine classifier to assess the discriminative power of the reduced feature sets in real-world applications.

Acknowledgement of Sources

All the theoretical results and methodologies discussed in this report are based on the seminal works of *Zou et al. (2006)* on Sparse Principal Component Analysis [1] and *Jenatton et al. (2010)* on Structured Sparse Principal Component Analysis [2]. These papers provide the mathematical foundation and optimization techniques that are central to our analysis

The results presented in this report, including mathematical formulations, optimization strategies, and comparisons between PCA, SPCA, and SSPCA, are directly derived from these pivotal studies.

2 Exploratory Data Analysis (EDA)

The exploratory data analysis focuses on understanding the structure and characteristics of the **Labeled Faces in the Wild (LFW)** dataset, which is used for this project. Below are the key findings and visualizations:

2.1 Dataset Overview

The dataset contains grayscale images of seven well-known individuals, with the following details:

- **Total Samples**: 1,288 images.
- **Image Dimensions**: Each image is resized to 50×37 pixels.
- **Number of Classes**: 7 unique individuals.
- **Classes**: Ariel Sharon, Colin Powell, Donald Rumsfeld, George W. Bush, Gerhard Schroeder, Hugo Chavez, Tony Blair

2.2 Sample Images

To gain a better understanding of the dataset, we visualized a subset of the images, showcasing examples for each class. The images reveal diverse poses, lighting conditions, and facial expressions, reflecting real-world variability.

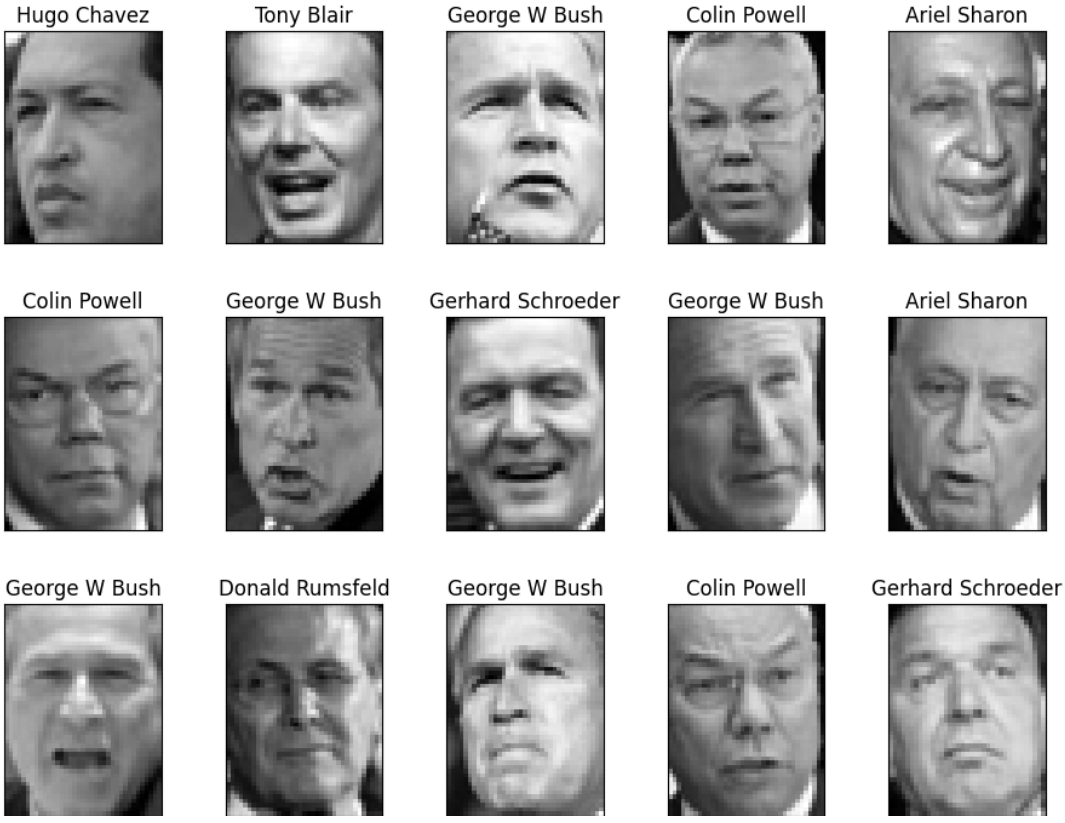

Figure 1: Sample images from the LFW dataset, labeled with the corresponding class names.

2.3 Pixel Intensity Distribution

The pixel intensity values, ranging from 0 (black) to 1 (white), were analyzed to understand their distribution across the dataset. The histogram shows a concentration of pixel intensities around mid-range values, with fewer extremely dark or bright pixels. This suggests that the dataset predominantly contains well-lit facial images.

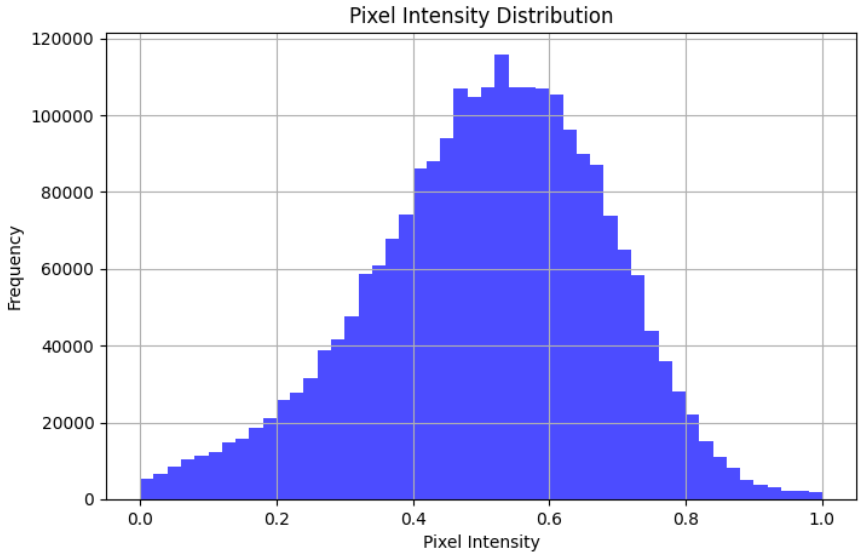

Figure 2: Histogram of pixel intensity distribution across all images in the dataset.

2.4 Class Distribution

The class distribution was analyzed to determine the number of samples for each individual. The dataset exhibits a slight imbalance, with *George W. Bush* having the highest number of samples, reflecting his prominence in the dataset. This imbalance could influence model performance and needs consideration during evaluation.

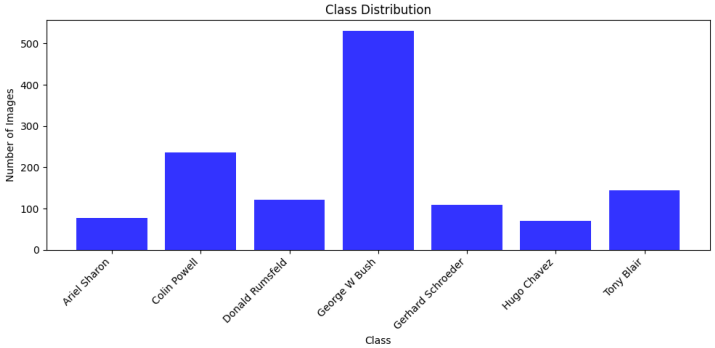

Figure 3: Bar chart showing the number of images per individual in the dataset.

2.5 Image Statistics

The mean and standard deviation of the pixel intensities were computed across all images:

- **Mean Image**: The average face computed across all samples highlights prominent facial features shared among the images, such as eyes and mouth regions.
- **Standard Deviation Image**: The variability image shows areas of high variance, particularly around regions like hairlines and facial contours, which differ significantly between individuals.

Figure 4: (Left) Mean image (average face). (Right) Standard deviation image (variability across all images).

2.6 Image Flattening and Dataset Representation

In this section, we describe how facial images of dimensions 50×37 pixels are flattened into 1-dimensional vectors of size 1850. Flattening is a common preprocessing step where a 2D matrix of pixel intensities is reshaped into a single row vector for compatibility with machine learning models.

Flattening a Single Image

An image $\mathbf{I}$ of size 50×37 (height $\times$ width) can be represented as a matrix:

$$\mathbf{I} = \begin{bmatrix} i_{1,1} & i_{1,2} & \cdots & i_{1,37} \\ i_{2,1} & i_{2,2} & \cdots & i_{2,37} \\ \vdots & \vdots & \ddots & \vdots \\ i_{50,1} & i_{50,2} & \cdots & i_{50,37} \end{bmatrix},$$

where each element $i_{h,w}$ represents the pixel intensity at row h and column w with values normalized between 0 (black) and 1 (white).

To flatten the image, the rows of the matrix are concatenated into a single row vector of size 1,850:

$$\mathbf{I}_{\text{flat}} = \left[i_{1,1}, i_{1,2}, \ldots, i_{1,37}, i_{2,1}, \ldots, i_{50,37}\right]$$

This transformation enables the representation of each image as a vector that can be stacked into a larger dataset.

Representation of the Dataset Matrix

Given a dataset of $1,288$ images, the flattened vectors are stacked into a single matrix $\mathbf{X}$, where:

$$\mathbf{X} = \begin{bmatrix} \mathbf{I}_{\text{flat},1} \\ \mathbf{I}_{\text{flat},2} \\ \vdots \\ \mathbf{i}_{\text{flat},1288} \end{bmatrix},$$

with $\mathbf{X}$ having the shape $1,288 \times 1,850$. Thus we have $\mathbf{X} \in \mathbb{R}^{1,288 \times 1,850}$

Additionally, the response vector $\mathbf{Y}$, which represents the class labels for the images, is of size $1,288$, with each element belonging to one of the 7 distinct classes:

$$\mathbf{Y} = \left[y_1, y_2, \ldots, y_{1288}\right]^\top, \quad y_i \in \{1, 2, \ldots, 7\}.$$

Importance of Flattening

Flattening allows images to be processed as feature vectors, facilitating operations such as dimensionality reduction, classification, and clustering. However, this operation also removes the spatial structure of the image, which can be reintroduced through specialized techniques like convolutional neural networks (CNNs) or locality-preserving transforms.

2.7 Key Insights

1. The dataset's diversity in lighting, poses, and expressions adds complexity to feature extraction tasks.
2. The slight imbalance in class distribution should be considered during model training and evaluation.
3. The pixel intensity distribution and image statistics provide valuable information for preprocessing and feature extraction.

This analysis sets the stage for applying dimensionality reduction techniques like PCA and SSPCA to extract meaningful facial features from the dataset.

3 Methodology of SPCA

3.1 PCA

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique. In the context of human face recognition, PCA is instrumental in extracting and encoding the most significant features of facial images. Facial images, often represented as high-dimensional matrices, can be processed by PCA to reduce their dimensions while retaining the essential patterns for classification or recognition tasks.

PCA operates by finding linear combinations of the original variables (pixel intensities, in the case of images) to form new uncorrelated variables called Principal Components (PCs). These PCs capture the maximum variance in the data, ensuring that most of the meaningful information is preserved while discarding noise and redundancy.

To compute PCA, we leverage the Singular Value Decomposition (SVD) of the data matrix $\mathbf{X}$. Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ represent a matrix where rows correspond to observations (e.g., facial images) and columns to features (e.g., pixel intensities). The SVD of $\mathbf{X}$ is expressed as:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$$

Here:

- $\mathbf{U} \in \mathbb{R}^{n \times n}$ contains the left singular vectors, which represent the principal components (PCs) scaled by their contributions.
- $\mathbf{D} \in \mathbb{R}^{n \times p}$ is a diagonal matrix of singular values, representing the variance captured by each PC.
- $\mathbf{V} \in \mathbb{R}^{p \times p}$ contains the right singular vectors, often referred to as the loadings of the PCs.

In human face recognition, the PCs act as the "eigenfaces," which are used to encode facial images into low-dimensional feature spaces. By retaining only a few leading PCs, we achieve dimensionality reduction, allowing for efficient recognition.

Despite its effectiveness, PCA has notable drawbacks:

- **Lack of Sparsity:** Each PC is a linear combination of all original variables (features). For facial images, this means that every pixel contributes to the PCs, making them challenging to interpret.
- **Interpretability Issues:** Since the loadings (components of $\mathbf{V}$) are dense and not explicitly associated with specific features, it is difficult to associate PCs with distinct regions or patterns in the image.
- **Potential Information Loss:** While PCA retains maximal variance, it does so without explicitly considering which features are most relevant to the task at hand.

3.2 The Lasso and the Elastic Net

Understanding the Lasso and Elastic Net is crucial for Sparse Principal Component Analysis (SPCA) because SPCA relies on introducing sparsity into principal components through regularization techniques. The Lasso provides a mechanism to enforce sparsity by shrinking coefficients to zero, while the Elastic Net improves upon this by handling correlated features effectively.

Let $\mathbf{Y} \in \mathbb{R}^n$ represent the response vector and $\mathbf{X} \in \mathbb{R}^{n \times p}$ the design matrix, where n is the number of observations and p is the number of predictors. Each column $\mathbf{X}_j$ corresponds to the j-th predictor. The vector $\boldsymbol{\beta} \in \mathbb{R}^p$ contains the regression coefficients.

The objective is to estimate $\boldsymbol{\beta}$ such that the model:

$$\mathbf{Y} \approx \mathbf{X}\boldsymbol{\beta},$$

minimizes the residual error while incorporating regularization to improve sparsity and generalization. The regularization parameters λ_1 and λ_2 control the strength of penalization.

Lasso

The Lasso (**Least Absolute Shrinkage and Selection Operator**) is a regularization method that adds an L_1 penalty to the regression coefficients, encouraging sparsity. It solves:

$$\hat{\boldsymbol{\beta}}_{\text{lasso}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j|,$$

where $\lambda_1 \geq 0$ controls the trade-off between model fit and sparsity. The Lasso works by shrinking some coefficients β_j to exactly zero, effectively selecting a subset of predictors. However, it struggles with highly correlated predictors and in high-dimensional settings ($p > n$).

Elastic Net

The **Elastic Net** addresses these limitations by combining the L_1 penalty of the Lasso with the L_2 penalty of Ridge Regression:

$$\hat{\boldsymbol{\beta}}_{\text{en}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda_2 \sum_{j=1}^{p} \beta_j^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j|.$$

Here:

- λ_1 enforces sparsity by shrinking coefficients to zero (Lasso effect).
- λ_2 stabilizes the solution by penalizing large coefficients (Ridge effect).

The Elastic Net is particularly useful for high-dimensional datasets ($p > n$) and when predictors are highly correlated, as it can include groups of correlated variables in the

model. It balances sparsity and model stability, making it a robust alternative to the Lasso.

By incorporating these regularization techniques, Sparse PCA overcomes the interpretability issues of traditional PCA by producing sparse loadings, focusing on the most important features in the data.

3.3 Sparse PCA

Sparse PCA (SPCA), as introduced by *Zou et al. (2006)* [1], addresses the interpretability issues of PCA by introducing sparsity into the loadings matrix $\mathbf{V}$. Sparsity ensures that only a subset of features contributes to each principal component (PC), making them more interpretable. SPCA achieves this by framing the PCA problem as a regularized optimization task, where sparsity-inducing penalties (such as the Lasso or Elastic Net) are applied.

In human face recognition, SPCA enables the extraction of features that correspond to localized regions of the face, such as eyes, nose, or mouth, rather than global patterns involving all pixels. This is particularly useful when the goal is to identify key facial features or reduce the number of explicitly used variables, enhancing both computational efficiency and interpretability.

PCA as a Regression Problem

Each PC in PCA is a linear combination of the p variables, which means its loadings can be recovered by solving a regression problem. Denoting the i-th PC as $Z_i = \mathbf{U}_i D_{ii}$, the regression problem is formulated as:

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Z}_i - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2.$$

Here:

- $\mathbf{X} \in \mathbb{R}^{n\times p}$ is the data matrix.
- $\mathbf{Z}_i$ is the i-th principal component.
- $\lambda > 0$ is the ridge penalty ensuring numerical stability and a unique solution, particularly when $p > n$ or when $\mathbf{X}$ is rank-deficient.

The normalized ridge solution is:

$$\hat{\mathbf{v}} = \frac{\hat{\boldsymbol{\beta}}_{\text{ridge}}}{\|\hat{\boldsymbol{\beta}}_{\text{ridge}}\|}.$$

This approach bridges the connection between PCA and regression. However, to achieve sparsity in the loadings, an L_1 penalty can be added to the optimization problem:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Z}_i - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda_2\|\boldsymbol{\beta}\|^2 + \lambda_1\|\boldsymbol{\beta}\|_1.$$

The L_1 penalty (Lasso) encourages sparsity, while the L_2 penalty (Ridge) stabilizes the solution. The resulting sparse loading vector $\hat{\mathbf{v}}_i$ is normalized, and the sparse principal component is given by:

$$\mathbf{X}\hat{\mathbf{v}}_i.$$

SPCA Criterion and Optimization

For multiple PCs, a self-contained regression-type criterion is introduced. Let $\mathbf{x}_i$ be the i-th row of the matrix $\mathbf{X}$, and let $\mathbf{A}$ and $\mathbf{B}$ (corresponding to $\mathbf{UD}$ and $\mathbf{V}$ in standard PCA) represent the loading and coefficient matrices, respectively.

The optimization problem is:

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}) = \arg\min_{\mathbf{A},\mathbf{B}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{A}\mathbf{B}^\top\mathbf{x}_i\|^2 + \sum_{j=1}^{k} \left(\lambda_1\|\boldsymbol{\beta}_j\|_1 + \lambda_2\|\boldsymbol{\beta}_j\|^2\right),$$

subject to $\mathbf{A}^\top\mathbf{A} = \mathbf{I}_{k\times k}$, where:

- $\mathbf{A} \in \mathbb{R}^{p\times k}$ contains the k sparse loading vectors.
- $\mathbf{B} \in \mathbb{R}^{n\times k}$ contains the associated coefficients for each principal component.
- λ_1 controls the sparsity (Lasso penalty). A larger λ_1 increases the number of coefficients equal to zero, thereby reducing the number of variables in the component.
- λ_2 controls stability (Ridge penalty).

This criterion allows us to compute sparse principal components in a flexible manner. Unlike traditional PCA, which computes PCs as dense linear combinations of all variables, SPCA enforces sparsity, making it possible to interpret the PCs in terms of specific features.

Key Properties and Advantages of SPCA

The advantages of SPCA include:

- **Localized Feature Extraction:** In face recognition, SPCA identifies critical regions (e.g., eyes, nose) rather than global patterns, improving interpretability.

- **Dimensionality Reduction with Sparsity:** SPCA reduces dimensions while retaining the most important features, enhancing computational efficiency.
- **Connection to Regression:** SPCA effectively uses regression techniques to approximate PCs, allowing for flexible sparsity constraints.

In summary, SPCA combines the power of PCA for dimensionality reduction with the interpretability and feature selection benefits of regularization techniques. This makes it particularly useful in high-dimensional settings like human face recognition, where extracting sparse, meaningful features is critical.

3.4 SSPCA

Structured Sparse Principal Component Analysis (SSPCA), as introduced by *Jenatton et al. (2010)* on Structured Sparse Principal Component Analysis [2], extends Sparse PCA (SPCA) by incorporating domain-specific structures into the sparsity constraints of principal components. Unlike SPCA, which enforces sparsity independently across variables, SSPCA introduces structured sparsity that aligns with inherent relationships or patterns in the data. This enhancement makes SSPCA particularly effective for high-dimensional datasets with spatial or temporal structure, such as facial image data.

Optimization Problem

The SSPCA optimization problem modifies the PCA framework by introducing a structured sparsity-inducing regularization term. Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ represent the data matrix, where n is the number of observations and p the number of features. SSPCA solves the following optimization problem:

$$\min_{\mathbf{U},\mathbf{V}} \|\mathbf{X} - \mathbf{U}\mathbf{V}^\top\|_F^2 + \lambda \sum_{g \in \mathcal{G}} \Omega_g(\mathbf{V}),$$

where:

- $\mathbf{U} \in \mathbb{R}^{n \times k}$: The projected data (scores).
- $\mathbf{V} \in \mathbb{R}^{p \times k}$: The sparse principal components (loadings).
- $\mathcal{G}$: A predefined set of groups or structures (e.g., pixel neighborhoods, spatial regions).
- $\Omega_g(\mathbf{V})$: A structured sparsity penalty, often the mixed ℓ_1/ℓ_2-norm.

The regularization term $\Omega_g(\mathbf{V})$ enforces sparsity over groups of variables, encouraging entire groups to be zeroed out when their contribution is minimal:

$$\Omega_g(\mathbf{V}) = \sum_{g \in \mathcal{G}} \|\mathbf{V}_g\|_2,$$

where $\mathbf{V}_g$ represents the loadings corresponding to group g. This penalty ensures that the extracted components align with meaningful structures in the data.

Optimization Strategy

Solving the SSPCA optimization problem involves alternating updates:

1. **Update U:** With $\mathbf{V}$ fixed, compute $\mathbf{U}$ using least squares:

$$\mathbf{U} = \mathbf{X}\mathbf{V}.$$

2. **Update V:** With $\mathbf{U}$ fixed, enforce structured sparsity on $\mathbf{V}$ using soft-thresholding or block-coordinate descent:

$$\mathbf{V}_g \leftarrow \max\left(0, \|\mathbf{V}_g\|_2 - \lambda\right) \cdot \frac{\mathbf{V}_g}{\|\mathbf{V}_g\|_2}, \quad \forall g \in \mathcal{G}.$$

Applications of SSPCA

SSPCA is valuable in various contexts where features exhibit inherent structure. Notable applications include:

- **Image Analysis:** In facial image recognition, SSPCA enforces sparsity over contiguous pixel regions, leading to interpretable components that highlight localized facial features such as eyes, nose, and mouth.
- **Genomics:** SSPCA identifies gene pathways or groups of correlated genes associated with specific traits, ensuring biological relevance.
- **Dimensionality Reduction:** SSPCA retains critical features while reducing dimensionality, making it suitable for high-dimensional datasets with structured relationships.

Advantages of SSPCA

The benefits of SSPCA include:

- **Improved Interpretability:** Structured sparsity ensures that principal components correspond to meaningful subsets of variables or regions.
- **Robustness:** SSPCA is less sensitive to noise and irrelevant variables by focusing on structured groups of features.
- **Flexibility:** Custom structures tailored to the dataset make SSPCA adaptable to diverse applications.

Results and Conclusion

For facial image datasets, SSPCA components correspond to interpretable regions of the face, such as the eyes and mouth. The structured sparsity constraints improve both interpretability and reconstruction performance compared to PCA and SPCA. SSPCA provides a robust framework for analyzing high-dimensional structured data, bridging the gap between dimensionality reduction and feature selection.

4 Application to Human-face Recognition

In this section, we apply PCA and Sparse PCA to the **Labeled Faces in the Wild (LFW)** dataset to evaluate their effectiveness in facial image analysis. The objective is to extract lower-dimensional representations that preserve discriminative information, enabling efficient and interpretable face recognition. Additionally, we compare the impact of these dimensionality reduction methods on predictive performance by utilizing a Support Vector Machine (SVM) classifier. This analysis sheds light on the trade-offs between accuracy, interpretability, and computational efficiency.

4.1 Application of PCA

We apply Principal Component Analysis (PCA) to the LFW dataset to reduce its dimensionality while retaining most of the variance. The primary goal of PCA here is to simplify the data structure, extract meaningful features, and visualize the dataset in a lower-dimensional space.

Dimensionality Reduction

After standardizing the dataset, PCA was performed with the aim of retaining 95% of the variance. The original dataset with 1,850 features was successfully reduced to 171 principal components, significantly reducing the dimensionality while maintaining most of the meaningful information:

- **Shape of Original Dataset (Standardized)**: $1,288 \times 1,850$
- **Shape of Reduced Dataset (PCA)**: $1,288 \times 171$

Visualizing Principal Components

The first five PCA components (eigenfaces) were visualized to understand the patterns they capture. These components represent directions in the feature space that explain the most variance in the dataset. Each principal component corresponds to a "weighted combination" of original pixel intensities, with brighter and darker regions highlighting areas of variation.

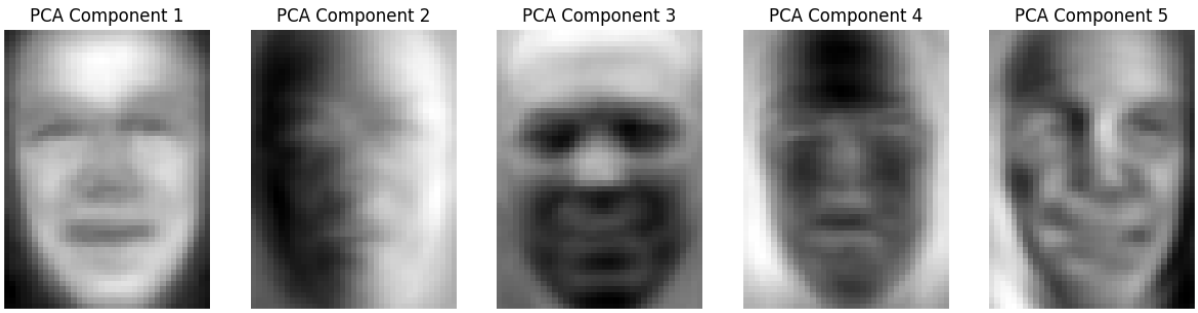

Figure 5: The first five PCA components (eigenfaces). These components highlight key patterns in the facial data, with lighter and darker areas corresponding to regions of high and low variance, respectively.

- **PCA Component 1**: Captures global brightness and symmetry across the face, emphasizing overall facial structure.
- **PCA Component 2**: Highlights asymmetry in facial features, such as lighting differences between the left and right sides of the face.
- **PCA Component 3**: Focuses on shadows and variations around the mouth and chin regions.
- **PCA Component 4**: Emphasizes variations in the eye and brow regions, possibly reflecting different expressions or lighting.
- **PCA Component 5**: Reflects changes in the cheek and mouth areas, such as variations in facial expressions.

These components, often called eigenfaces, are critical in understanding patterns in facial data. They demonstrate PCA's ability to reduce dimensionality while preserving key features. However, the lack of localized interpretability motivates the use of methods such as Sparse PCA.

Insights and Observations

- The reduced dimensionality (171 components) retains 95% of the variance, significantly simplifying the dataset.
- The PCA components (eigenfaces) provide interpretable patterns that correspond to key areas of variation in facial features.

This analysis demonstrates PCA's ability to reduce dimensionality effectively while preserving key patterns in the data. However, the lack of sparsity in PCA components motivates the exploration of Sparse PCA (SPCA) to improve interpretability.

4.2 Application of Sparse PCA

Sparse Principal Component Analysis (Sparse PCA) was applied to the LFW dataset to address the limitations of traditional PCA, particularly the lack of interpretability due to dense principal components. By enforcing sparsity through regularization, Sparse PCA identifies principal components where only a subset of features contributes, making the results more interpretable and localized.

Similar to PCA, Sparse PCA was used to reduce the dimensionality of the dataset while retaining meaningful variance. Sparse PCA successfully reduced the original dataset from 1,850 features to 171 sparse components:

Visualizing Sparse PCA Components

The first five Sparse PCA components are visualized below. Unlike PCA, these components demonstrate clear sparsity, highlighting specific localized regions of the face (e.g., eyes, mouth, and other distinct facial features). This sparsity enhances interpretability, as each component captures distinct and meaningful patterns in the data.

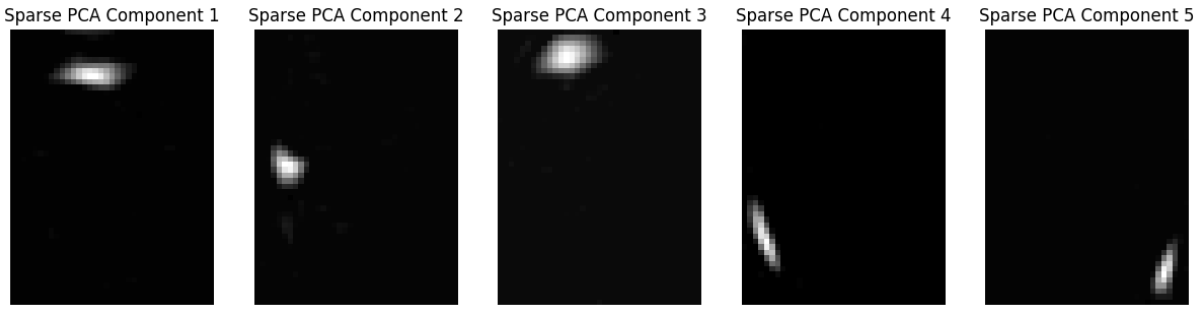

Figure 6: Visualization of the first five Sparse PCA components. Each component highlights localized regions of variation, such as specific facial features, making them more interpretable compared to dense PCA components.

- **Sparse PCA Component 1**: Highlights a localized region, likely corresponding to an eye or part of the forehead.
- **Sparse PCA Component 2**: Focuses on a separate facial feature, such as the other eye or a shadowed region.
- **Sparse PCA Component 3**: Captures a distinct upper facial feature, potentially related to the eyes or forehead region.
- **Sparse PCA Component 4**: Shifts attention to the lower face, possibly highlighting the mouth or jawline, reflecting subtle variations in expressions or facial structure.
- **Sparse PCA Component 5**: Isolates another localized region, potentially the cheeks or another distinct facial feature.

These sparse components demonstrate how Sparse PCA isolates specific facial features, offering improved interpretability compared to traditional PCA. This makes Sparse PCA particularly useful for tasks requiring feature selection and localization, such as in our case facial recognition.

Comparison Between PCA and Sparse PCA

- **Component Structure:** PCA components are dense and global, capturing variance across the entire image, while Sparse PCA components are sparse and localized, highlighting specific regions such as eyes, nose, or mouth.
- **Interpretability:** Sparse PCA components are easier to interpret, as they focus on distinct regions of the face, whereas PCA components are harder to relate to specific features.

4.3 Prediction model

To compare the different methods we use, we will do a prediction model. We decide to use a Support Vector Machine (SVM) because it works well for high-dimensional data.

Support Vector Machine

SVM are supervised learning algorithms used for classification and regression tasks. The main idea of SVM is to find a hyperplane that best separates data points from different classes while maximizing the margin, which is the distance between the hyperplane and the nearest data points from each class (called support vectors). This maximized margin helps the model generalize better to unseen data.

Mathematically, SVM solves the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \, \forall i$$

where $\mathbf{w}$ is the normal vector to the hyperplane, b is the bias term, $\mathbf{x}_i$ are feature vectors, and y_i are class labels (the name of the person in our dataset, an integer between 1 and 7). For non-linearly separable data, slack variables ξ_i are introduced to allow misclassifications, controlled by the regularization parameter C, which balances the trade-off between maximizing the margin and minimizing errors.

For datasets that are not linearly separable, SVM uses the kernel trick to map data into a higher-dimensional space where a linear hyperplane can separate the classes. Common kernel functions include:

- **Linear**: Suitable for linearly separable data.
- **Radial Basis Function (RBF)**: Captures non-linear relationships effectively.

- **Polynomial**: Handles more complex decision boundaries.

Optimization of SVM

To optimize our SVM model, we will use a RBF kernel because it is even more suitable to high dimensional data than the linear and the polynomial kernel. There is two hyperparameters to tune:

- **C**: The regularization parameter that controls the trade-off between maximizing the margin and minimizing classification errors; a larger C reduces misclassification but may lead to overfitting.

- **Gamma**: Defines the influence of a single data point on the decision boundary; a smaller γ means far-reaching influence, while a larger γ focuses on closer points.

We performe hyperparameter tuning using grid search and cross-validation. The process can be summarized as follows:

- We defined a parameter grid to test:
  - Regularization parameter $C \in \{1, 5, 10, 20\}$.
  - Kernel coefficient $\gamma \in \{\text{scale}, 0.1, 0.01, 0.001\}$ for the RBF kernel (scale being the default value proposed by scikit-learn).
- A 5-fold cross-validation was applied to evaluate the performance of each combination of C and γ, using accuracy as the evaluation metric.
- The best hyperparameters were selected based on the highest cross-validation accuracy.

Results

Here are the results obtained after the grid search:

- **No PCA:** Achieved an accuracy of 0.845 with $C = 5$ and $\gamma = \text{scale}$.
- **PCA:** Achieved an accuracy of 0.845 with $C = 5$ and $\gamma = \text{scale}$.
- **Sparse PCA:** Achieved an accuracy of 0.841 with $C = 10$ and $\gamma = \text{scale}$.
- **MiniBatch Sparse PCA:** Achieved the highest accuracy of 0.853 with $C = 5$ and $\gamma = \text{scale}$.

More detaild results (with classification reports and confusion matrices) are avalaible in appendix.

The results show that PCA and SPCA yield slightly different accuracies, with PCA achieving 0.845 and SPCA achieving 0.841. Both methods have similar performance because the dataset likely contains information distributed across most features, which PCA captures effectively.

SPCA enforces sparsity by ignoring some features, which can lead to a slight loss of discriminative information when the data does not inherently support sparsity. This explains the minor drop in accuracy for SPCA.

Interestingly, MiniBatch SPCA achieves a slightly higher accuracy of 0.853. This may be due to its ability to balance sparsity with computational efficiency, preserving more of the discriminative structure in the data.

4.4 To go further

There are two points that could be explored further to potentially improve the results:

- **Applying SSPCA:** The Structured Sparse Principal Component Analysis (SSPCA) method could be applied to the dataset. SSPCA introduces additional structure to the sparsity pattern, which may better capture meaningful relationships between features in the data. Comparing the predictive accuracy of SSPCA with SPCA and Mini Batch SPCA would provide insight into its effectiveness.
- **Experimenting with Different Levels of Variance in PCA:** The current PCA implementation retains 95% of the variance. It would be interesting to try different levels of variance (e.g., 90%, 85%, or 99%) to observe how the dimensionality reduction affects the model's predictive accuracy. Reducing the retained variance might help eliminate noise, while increasing it could preserve more information and improve classification performance.

Exploring these directions may provide additional insights into optimizing feature extraction techniques and improving the overall model performance.

5 Conclusion and Perspectives

This project analyzed PCA and Sparse PCA for dimensionality reduction on the **Labeled Faces in the Wild (LFW)** dataset. PCA reduced the dataset to 171 components while retaining 95% of the variance, capturing global patterns but lacking interpretability. Sparse PCA introduced sparsity, highlighting localized facial features like eyes and mouth, but slightly reduced classification accuracy.

Key Insights:

- PCA is effective for retaining variance and computational efficiency but lacks interpretability.
- Sparse PCA improves feature interpretability by localizing components but may lose some discriminative power.
- MiniBatch Sparse PCA achieved the best accuracy (0.853), balancing sparsity and efficiency.

Future Directions:

- Explore **SSPCA** for improved accuracy and structured interpretability.
- Experiment with different variance thresholds in PCA (e.g., 90%, 99%).
- Integrate sparse features with deep learning models for hybrid approaches.
- Evaluate these methods on larger, more complex datasets.

Bibliography

References

[1] Zou, H., Hastie, T., & Tibshirani, R. (2006). Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, *15*(2), 265-286. https://doi.org/10.1198/106186006X113430

[2] Jenatton, R., Obozinski, G., & Bach, F. (2010, March). Structured sparse principal component analysis. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 366-373). JMLR Workshop and Conference Proceedings. http://proceedings.mlr.press/v9/jenatton10a/jenatton10a.pdf

A Appendix: Python Code

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_lfw_people
import os

figures_dir = "../figures"
os.makedirs(figures_dir, exist_ok=True)

# Step 1: Load the dataset
print("Loading LFW dataset...")
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
images = lfw_dataset.images
X = lfw_dataset.data
n_samples, h, w = images.shape
target_names = lfw_dataset.target_names
n_classes = len(target_names)

print(f"Dataset loaded with {n_samples} samples.")
print(f"Image dimensions: {h}x{w}")
print(f"Number of classes: {n_classes}")
print("Classes:", target_names)

# Step 2: Visualize a few sample images
def plot_sample_images(images, target, target_names, h, w, n_row=3,
    n_col=5):
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace
        =.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(target_names[target[i]], size=12)
        plt.xticks(())
        plt.yticks(())
    save_path = os.path.join(figures_dir, "sample_images.png")
    plt.savefig(save_path)
    plt.show()

print("Displaying sample images...")
plot_sample_images(images, lfw_dataset.target, target_names, h, w)

# Step 3: Analyze pixel intensity distribution
def plot_pixel_distribution(images):
    pixel_values = images.flatten()
    plt.figure(figsize=(8, 5))
    plt.hist(pixel_values, bins=50, color='blue', alpha=0.7)
    plt.title("Pixel Intensity Distribution")
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    plt.grid(True)
    save_path = os.path.join(figures_dir, "pixel_intensity.png")
    plt.savefig(save_path)
    plt.show()
```

```
52
53 print("Analyzing␣pixel␣intensity␣distribution...")
54 plot_pixel_distribution(images)
55
56 # Step 4: Class distribution
57 def plot_class_distribution(target, target_names):
58     class_counts = np.bincount(target)
59     plt.figure(figsize=(10, 5))
60     plt.bar(range(len(target_names)), class_counts, color='blue', alpha
       =0.8)
61     plt.title("Class␣Distribution")
62     plt.xlabel("Class")
63     plt.ylabel("Number␣of␣Images")
64     plt.xticks(range(len(target_names)), target_names, rotation=45, ha="
       right")
65     plt.tight_layout()
66     save_path = os.path.join(figures_dir, "class_distribution.png")
67     plt.savefig(save_path)
68     plt.show()
69
70 print("Analyzing␣class␣distribution...")
71 plot_class_distribution(lfw_dataset.target, target_names)
72
73 # Step 5: Summary statistics for images
74 def compute_image_statistics(images):
75     mean_image = np.mean(images, axis=0)
76     std_image = np.std(images, axis=0)
77     return mean_image, std_image
78
79 mean_image, std_image = compute_image_statistics(X)
80
81 # Visualize mean and standard deviation images
82 def plot_image_statistics(mean_image, std_image, h, w):
83     plt.figure(figsize=(10, 5))
84     plt.subplot(1, 2, 1)
85     plt.imshow(mean_image.reshape((h, w)), cmap=plt.cm.gray)
86     plt.title("Mean␣Image")
87     plt.axis("off")
88     plt.subplot(1, 2, 2)
89     plt.imshow(std_image.reshape((h, w)), cmap=plt.cm.gray)
90     plt.title("Standard␣Deviation␣Image")
91     plt.axis("off")
92     plt.tight_layout()
93     save_path = os.path.join(figures_dir, "mean_sd.png")
94     plt.savefig(save_path)
95     plt.show()
96
97 print("Visualizing␣mean␣and␣standard␣deviation␣of␣images...")
98 plot_image_statistics(mean_image, std_image, h, w)
```

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import fetch_lfw_people
4 from sklearn.decomposition import PCA, MiniBatchSparsePCA
5 from sklearn.preprocessing import StandardScaler
6
```

```
# Step 1: Load the LFW dataset
print("Loading LFW dataset...")
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = lfw_dataset.data
n_samples, h, w = lfw_dataset.images.shape

print(f"Dataset loaded with {n_samples} images of size {h}x{w}.")

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 2: PCA Implementation
print("Performing PCA...")
n_components = 15
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)
pca_components = pca.components_

# Step 3: Custom SSPCA Implementation
class SSPCA:
    def __init__(self, n_components, alpha=0.1, max_iter=50, group_size
        =5, tol=1e-6, verbose=False):
        self.n_components = n_components
        self.alpha = alpha
        self.max_iter = max_iter
        self.group_size = group_size
        self.tol = tol
        self.verbose = verbose

    def fit_transform(self, X):
        pca = PCA(n_components=self.n_components)
        Z = pca.fit_transform(X)  # PCA initialization
        components = pca.components_
        for iteration in range(self.max_iter):
            old_components = components.copy()
            components = self._apply_sparsity(components)
            Z = X @ components.T  # Update latent representation
            components = (Z.T @ X) / (np.linalg.norm(Z.T @ X, axis=1,
                keepdims=True) + 1e-8)

            # Check for convergence
            diff = np.linalg.norm(components - old_components)
            if self.verbose:
                print(f"Iteration {iteration + 1}, component diff = {
                    diff}")
            if diff < self.tol:
                if self.verbose:
                    print("Converged early.")
                break
        self.components_ = components
        return Z

    def _apply_sparsity(self, components):
        for i in range(components.shape[0]):  # For each principal
            component
```

```
            for start in range(0, components.shape[1], self.group_size):
                end = min(start + self.group_size, components.shape[1])
                group = components[i, start:end]
                group_norm = np.linalg.norm(group)
                if group_norm <= self.alpha:
                    # Zero out the entire group
                    components[i, start:end] = 0.0
                else:
                    # Shrink the group
                    shrink_factor = 1 - self.alpha / group_norm
                    components[i, start:end] = group * shrink_factor
        return components

print("Performing personalized SSPCA...")
sspca = SSPCA(n_components=n_components, alpha=0.1)
X_sspca = sspca.fit_transform(X_scaled)
sspca_components = sspca.components_

print("Performing Mini Batch SSPCA...")
mb_sspca = MiniBatchSparsePCA(n_components=n_components, alpha=0.1,
    batch_size=100, max_iter=50)
X_mb_sspca = mb_sspca.fit_transform(X_scaled)
mb_sspca_components = mb_sspca.components_

# Step 4: Visualization of Components
def plot_gallery(title, images, n_col=5, n_row=3, image_shape=(h, w)):
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.suptitle(title, size=16)
    for i, comp in enumerate(images[:n_col * n_row]):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(comp.reshape(image_shape), cmap=plt.cm.gray)
        plt.xticks(())
        plt.yticks(())
    plt.show()

print("Visualizing Original Images...")
plot_gallery("Original Images", X)

print("Visualizing PCA components...")
plot_gallery("PCA Components", pca_components, image_shape=(h, w))

print("Visualizing personalized SSPCA components...")
plot_gallery("SSPCA Components", sspca_components, image_shape=(h, w))

print("Visualizing Mini Batch SSPCA components...")
plot_gallery("Mini Batch SSPCA Components", mb_sspca_components,
    image_shape=(h, w))

# Step 5: Reconstruction Error Comparison
X_pca_reconstructed = pca.inverse_transform(X_pca)
X_sspca_reconstructed = X_scaled @ sspca_components.T @ sspca_components
X_mb_sspca_reconstructed = X_scaled @ mb_sspca_components.T @
    mb_sspca_components

pca_reconstruction_error = np.mean((X_scaled - X_pca_reconstructed) **
    2)
```

```
sspca_reconstruction_error = np.mean((X_scaled - X_sspca_reconstructed)
    ** 2)
mb_sspca_reconstruction_error = np.mean((X_scaled -
    X_mb_sspca_reconstructed) ** 2)

print(f"PCA Reconstruction Error: {pca_reconstruction_error:.4f}")
print(f"Personalized SSPCA Reconstruction Error: {
    sspca_reconstruction_error:.4f}")
print(f"Mini Batch SSPCA Reconstruction Error: {
    mb_sspca_reconstruction_error:.4f}")
```

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA, SparsePCA, MiniBatchSparsePCA
from sklearn.preprocessing import StandardScaler
from eda import X

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Performing PCA...")
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)
pca_components = pca.components_

print(f"Shape of X_scaled: {X_scaled.shape}")  # Standardized features
print(f"Shape of X_pca: {X_pca.shape}")  # Reduced features

# Define Sparse PCA with desired number of components
n_components = 171
sparse_pca = SparsePCA(n_components=n_components, random_state=42, alpha
    =1)

# Fit and transform the data
X_sparse_pca = sparse_pca.fit_transform(X_scaled)
sparse_pca_components = sparse_pca.components_

# Print shapes
print(f"Shape of X_scaled: {X_scaled.shape}")  # Standardized features
print(f"Shape of X_sparse_pca: {X_sparse_pca.shape}")  # Reduced
    features

# Define Mini Batch Sparse PCA with desired number of components

mbsparse_pca = MiniBatchSparsePCA(n_components=n_components, alpha=0.1,
    batch_size=100, max_iter=50, random_state=42)

# Fit and transform the data
X_mbsparse_pca = mbsparse_pca.fit_transform(X_scaled)
mbsparse_pca_components = mbsparse_pca.components_

# Print shapes
print(f"Shape of X_scaled: {X_scaled.shape}")  # Standardized features
print(f"Shape of X_sparse_pca: {X_mbsparse_pca.shape}")  # Reduced
    features
```

```
42
43  # Visualize the first few PCA components as images
44  n_visualize = 5  # Number of components to visualize
45  plt.figure(figsize=(15, 5))
46
47  for i in range(n_visualize):
48      plt.subplot(1, n_visualize, i + 1)
49      plt.imshow(pca_components[i].reshape(h, w), cmap='gray')
50      plt.title(f"PCA Component {i+1}")
51      plt.axis('off')
52
53  plt.show()
54
55  n_visualize = 5  # Number of components to visualize
56  plt.figure(figsize=(15, 5))
57
58  for i in range(n_visualize):
59      plt.subplot(1, n_visualize, i + 1)
60      plt.imshow(sparse_pca_components[i].reshape(h, w), cmap='gray')
61      plt.title(f"Sparse PCA Component {i+1}")
62      plt.axis('off')
63
64  plt.show()
65
66  n_visualize = 5  # Number of components to visualize
67  plt.figure(figsize=(15, 5))
68
69  for i in range(n_visualize):
70      plt.subplot(1, n_visualize, i + 1)
71      plt.imshow(mbsparse_pca_components[i].reshape(h, w), cmap='gray')
72      plt.title(f"Sparse PCA Component {i+1}")
73      plt.axis('off')
74
75  plt.show()
76
77  # 2D PCA visualization
78  plt.figure(figsize=(8, 6))
79  plt.scatter(X_pca[:, 0], X_pca[:, 1], c=lfw_dataset.target, cmap='
      rainbow', alpha=0.7, edgecolor='k')
80  plt.colorbar(ticks=range(len(target_names)), label="Class")
81  plt.title("PCA Projection (2D)")
82  plt.xlabel("PCA Component 1")
83  plt.ylabel("PCA Component 2")
84  plt.grid()
85  plt.show()
86
87  # 2D Sparse PCA visualization
88  plt.figure(figsize=(8, 6))
89  plt.scatter(X_sparse_pca[:, 0], X_sparse_pca[:, 1], c=lfw_dataset.target
      , cmap='rainbow', alpha=0.7, edgecolor='k')
90  plt.colorbar(ticks=range(len(target_names)), label="Class")
91  plt.title("Sparse PCA Projection (2D)")
92  plt.xlabel("Sparse PCA Component 1")
93  plt.ylabel("Sparse PCA Component 2")
94  plt.grid()
95  plt.show()
```

```

# 2D Mini-batch Sparse PCA visualization
plt.figure(figsize=(8, 6))
plt.scatter(X_mbsparse_pca[:, 0], X_mbsparse_pca[:, 1], c=lfw_dataset.target, cmap='rainbow', alpha=0.7, edgecolor='k')
plt.colorbar(ticks=range(len(target_names)), label="Class")
plt.title("Mini-batch Sparse PCA Projection (2D)")
plt.xlabel("Sparse PCA Component 1")
plt.ylabel("Sparse PCA Component 2")
plt.grid()
plt.show()

# 3D PCA visualization (if n_components >= 3)
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=lfw_dataset.target, cmap='rainbow', alpha=0.7)
plt.colorbar(scatter, label="Class")
ax.set_title("PCA Projection (3D)")
ax.set_xlabel("PCA Component 1")
ax.set_ylabel("PCA Component 2")
ax.set_zlabel("PCA Component 3")
plt.show()

# 3D Sparse PCA visualization (if n_components >= 3)
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_sparse_pca[:, 0], X_pca[:, 1], X_sparse_pca[:, 2], c=lfw_dataset.target, cmap='rainbow', alpha=0.7)
plt.colorbar(scatter, label="Class")
ax.set_title("SPCA Projection (3D)")
ax.set_xlabel("SPCA Component 1")
ax.set_ylabel("SPCA Component 2")
ax.set_zlabel("SPCA Component 3")
plt.show()

# 3D MB Sparse PCA visualization (if n_components >= 3)
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_mbsparse_pca[:, 0], X_pca[:, 1], X_mbsparse_pca[:, 2], c=lfw_dataset.target, cmap='rainbow', alpha=0.7)
plt.colorbar(scatter, label="Class")
ax.set_title("Mini-batch SPCA Projection (3D)")
ax.set_xlabel("Mini-batch SPCA Component 1")
ax.set_ylabel("Mini-batch SPCA Component 2")
ax.set_zlabel("Mini-batch SPCA Component 3")
plt.show()
```

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV
from eda import target_names, n_classes, lfw_dataset
from spca import X_scaled, pca, sparse_pca, mbsparse_pca


```

```
8
9  # Create a train set and a test set
10 X_train, X_test, y_train, y_test = train_test_split(
11     X_scaled, lfw_dataset.target, test_size=0.2, random_state=42,
       shuffle=True, stratify=None
12 )
13
14 # Transform the data using PCA
15 X_train_pca = pca.fit_transform(X_train)
16 X_test_pca = pca.transform(X_test)
17
18 # Transform the data using Sparse PCA
19 X_train_sparse_pca = sparse_pca.fit_transform(X_train)
20 X_test_sparse_pca = sparse_pca.transform(X_test)
21
22 # Transform the data using MiniBatchSparsePCA
23 X_train_mbsparse_pca = mbsparse_pca.fit_transform(X_train)
24 X_test_mbsparse_pca = mbsparse_pca.transform(X_test)
25
26 # Define the parameter grid
27 param_grid = {
28     'C': [1, 5, 10, 20],  # Test various regularization strengths
29     'gamma': ['scale', 0.1, 0.01, 0.001]  # Test different gamma values
30 }
31
32 # Instantiate the SVC model
33 svm = SVC(kernel='rbf', random_state=42)
34
35 # Setup the GridSearchCV
36 grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5,
     scoring='accuracy', n_jobs=-1)
37
38 # Fit the GridSearchCV to the training data
39 grid_search.fit(X_train, y_train)
40
41 # Get the best parameters and the best score
42 best_params = grid_search.best_params_
43 best_score = grid_search.best_score_
44
45 print("Results for raw data:")
46
47 print(f'Best Parameters: {best_params}')
48 print(f'Best Cross-Validation Accuracy: {best_score:.4f}')
49
50 # Use the best parameters to make predictions on the test set
51 best_svm = grid_search.best_estimator_
52 y_pred = best_svm.predict(X_test)
53 acc = accuracy_score(y_test, y_pred)
54 print(f'Accuracy on test data with tuned parameters: {acc:.4f}')
55
56 # print classifiction results
57 print(classification_report(y_test, y_pred, target_names = target_names)
     )
58 # print confusion matrix
59 print("Confusion Matrix is:")
60 print(confusion_matrix(y_test, y_pred, labels = range(n_classes)))
```

```
61
62
63  grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5,
        scoring='accuracy', n_jobs=-1)
64
65  # Fit the GridSearchCV to the training data
66  grid_search.fit(X_train_pca, y_train)
67
68  # Get the best parameters and the best score
69  best_params = grid_search.best_params_
70  best_score = grid_search.best_score_
71
72  print("Results for PCA:")
73
74  print(f'Best Parameters: {best_params}')
75  print(f'Best Cross-Validation Accuracy: {best_score:.4f}')
76
77  # Use the best parameters to make predictions on the test set
78  best_svm = grid_search.best_estimator_
79  y_pred = best_svm.predict(X_test_pca)
80  acc = accuracy_score(y_test, y_pred)
81  print(f'Accuracy on test data with tuned parameters: {acc:.4f}')
82
83  # print classifiction results
84  print(classification_report(y_test, y_pred, target_names = target_names)
        )
85  # print confusion matrix
86  print("Confusion Matrix is:")
87  print(confusion_matrix(y_test, y_pred, labels = range(n_classes)))
88
89  grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5,
        scoring='accuracy', n_jobs=-1)
90
91  # Fit the GridSearchCV to the training data
92  grid_search.fit(X_train_sparse_pca, y_train)
93
94  # Get the best parameters and the best score
95  best_params = grid_search.best_params_
96  best_score = grid_search.best_score_
97
98  print("Results for SPCA:")
99
100 print(f'Best Parameters: {best_params}')
101 print(f'Best Cross-Validation Accuracy: {best_score:.4f}')
102
103 # Use the best parameters to make predictions on the test set
104 best_svm = grid_search.best_estimator_
105 y_pred = best_svm.predict(X_test_sparse_pca)
106 acc = accuracy_score(y_test, y_pred)
107 print(f'Accuracy on test data with tuned parameters: {acc:.4f}')
108
109 # print classifiction results
110 print(classification_report(y_test, y_pred, target_names = target_names)
        )
111 # print confusion matrix
112 print("Confusion Matrix is:")
```

```
print(confusion_matrix(y_test, y_pred, labels = range(n_classes)))

grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)

# Fit the GridSearchCV to the training data
grid_search.fit(X_train_mbsparse_pca, y_train)

# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Results for Mini Batch SPCA:")

print(f'Best Parameters: {best_params}')
print(f'Best Cross-Validation Accuracy: {best_score:.4f}')

# Use the best parameters to make predictions on the test set
best_svm = grid_search.best_estimator_
y_pred = best_svm.predict(X_test_mbsparse_pca)
acc = accuracy_score(y_test, y_pred)
print(f'Accuracy on test data with tuned parameters: {acc:.4f}')

# print classifiction results
print(classification_report(y_test, y_pred, target_names = target_names)
    )
# print confusion matrix
print("Confusion Matrix is:")
print(confusion_matrix(y_test, y_pred, labels = range(n_classes)))
```

B Detailed result of SVM

- **1** = Ariel Sharon
- **2** = Colin Powell
- **3** = Donald Rumsfeld
- **4** = George W Bush
- **5** = Gerhard Schroeder
- **6** = Hugo Chavez
- **7** = Tony Blair

Class	Precision	Recall	F1-Score	Support
1	0.89	0.73	0.80	11
2	0.80	0.91	0.85	47
3	0.93	0.64	0.76	22
4	0.84	0.96	0.90	119
5	0.75	0.79	0.77	19
6	1.00	0.38	0.56	13
7	0.95	0.70	0.81	27
Accuracy	**0.84 (Overall)**			
Macro Avg	0.88	0.73	0.78	258
Weighted Avg	0.86	0.84	0.84	258

Table 1: Classification report for raw data.

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	8	1	1	1	0	0	0
2	1	43	0	3	0	0	0
3	0	1	14	7	0	0	0
4	0	4	0	114	1	0	0
5	0	0	0	3	15	0	1
6	0	3	0	3	2	5	0
7	0	2	0	4	2	0	19

Table 2: Confusion matrix for raw data.

Class	Precision	Recall	F1-Score	Support
1	0.89	0.73	0.80	11
2	0.78	0.91	0.84	47
3	0.93	0.64	0.76	22
4	0.84	0.96	0.90	119
5	0.78	0.74	0.76	19
6	1.00	0.38	0.56	13
7	0.95	0.74	0.83	27
Accuracy	**0.84 (Overall)**			
Macro Avg	0.88	0.73	0.78	258
Weighted Avg	0.86	0.84	0.84	258

Table 3: Classification report for PCA.

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	8	1	1	1	0	0	0
2	1	43	0	3	0	0	0
3	0	2	14	6	0	0	0
4	0	4	0	114	1	0	0
5	0	1	0	3	14	0	1
6	0	3	0	4	1	5	0
7	0	1	0	4	2	0	20

Table 4: Confusion matrix for PCA.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
1	0.89	0.73	0.80	11
2	0.78	0.89	0.83	47
3	0.93	0.64	0.76	22
4	0.85	0.97	0.91	119
5	0.75	0.79	0.77	19
6	1.00	0.31	0.47	13
7	0.90	0.70	0.79	27
Accuracy	**0.84 (Overall)**			
Macro Avg	0.87	0.72	0.76	258
Weighted Avg	0.85	0.84	0.83	258

Table 5: Classification report for SPCA.

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	8	1	1	1	0	0	0
2	1	42	0	3	0	0	1
3	0	2	14	6	0	0	0
4	0	4	0	115	0	0	0
5	0	0	0	3	15	0	1
6	0	3	0	3	3	4	0
7	0	2	0	4	2	0	19

Table 6: Confusion matrix for SPCA.

Class	Precision	Recall	F1-Score	Support
1	0.86	0.55	0.67	11
2	0.83	0.96	0.89	47
3	0.88	0.64	0.74	22
4	0.84	0.96	0.90	119
5	0.79	0.79	0.79	19
6	1.00	0.38	0.56	13
7	0.95	0.78	0.86	27
Accuracy	**0.85 (Overall)**			
Macro Avg	0.88	0.72	0.77	258
Weighted Avg	0.86	0.85	0.84	258

Table 7: Classification report for Mini Batch SPCA.

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	6	1	2	2	0	0	0
2	1	45	0	1	0	0	0
3	0	1	14	7	0	0	0
4	0	4	0	114	1	0	0
5	0	0	0	3	15	0	1
6	0	3	0	5	0	5	0
7	0	0	0	3	3	0	21

Table 8: Confusion matrix for Mini Batch SPCA.