Code Documentation

Scatter1.py –

The Python script performs dimensionality reduction on a dataset by calculating the mixture scatter matrix and projecting the data onto a lower-dimensional space using eigenvalue decomposition. Specifically, it reduces the data to two dimensions and saves the reduced data to an output file.

Function Summaries:

1. `calculate\_mixture\_scatter(X)`:

This function computes the mixture scatter matrix for the dataset `X`. It begins by calculating the global mean vector of all samples in `X`. It then initializes a scatter matrix filled with zeros. For each sample in `X`, it computes the difference between the sample and the global mean, reshapes this difference into a column vector, and calculates the outer product of this vector with its transpose. This outer product represents the scatter contributed by that sample. The function accumulates these outer products for all samples to form the mixture scatter matrix, which encapsulates the total variance and covariance within the dataset.

2. `reduce\_dimensions(data, scatter\_matrix)`:

This function reduces the dimensionality of the input dataset `data` to two dimensions using the provided `scatter\_matrix`. It performs eigenvalue decomposition on the scatter matrix using `np.linalg.eigh`, obtaining the eigenvalues and corresponding eigenvectors. It sorts the eigenvalues in ascending order and selects the indices of the two smallest eigenvalues. The eigenvectors associated with these indices form the projection matrix. By multiplying the original data with this projection matrix, the function projects the data onto a new two-dimensional space, effectively reducing its dimensions while preserving certain characteristics captured by the scatter matrix.

3. `main()`:

This function serves as the entry point of the script and orchestrates the overall process. It first checks that the correct number of command-line arguments is provided, expecting a data file, a labels file, and an output file name. It then loads the dataset `X` from the specified data file and the labels `y` from the labels file. The function computes the mixture scatter matrix by calling `calculate\_mixture\_scatter(X)` and then reduces the dimensionality of `X` by calling `reduce\_dimensions(X, scatter\_matrix)`. Finally, it saves the reduced two-dimensional data to the specified output file and includes error handling to catch and report any exceptions that may occur during the execution.

Scatter2.py –

The Python script performs dimensionality reduction on a dataset by computing the mixture scatter matrix and projecting the data onto a lower-dimensional space using eigenvalue decomposition. The main goal is to reduce high-dimensional data to two dimensions, which is useful for visualization or simplifying further analysis. The script reads input data and labels from files, processes the data to compute the scatter matrix, reduces the dimensionality, and saves the resulting two-dimensional data to an output file.

Function Summaries:

1. `calculate\_mixture\_scatter(X)`:

This function computes the mixture scatter matrix for the dataset `X`. It calculates the global mean vector of all samples and initializes a scatter matrix filled with zeros. For each sample in `X`, it computes the difference between the sample and the global mean, reshapes this difference into a column vector, and adds the outer product of this vector with its transpose to the scatter matrix. The resulting mixture scatter matrix represents the total variability within the dataset, capturing how the data is spread around the global mean.

2. `reduce\_dimensions(data, scatter\_matrix)`:

This function reduces the dimensionality of the input `data` to two dimensions using the provided `scatter\_matrix`. It performs eigenvalue decomposition on the scatter matrix to obtain eigenvalues and corresponding eigenvectors. It sorts the eigenvalues and selects the eigenvectors associated with the eigenvalues. These selected eigenvectors form the projection matrix. By multiplying the original data with this projection matrix, the function projects the data onto a new two-dimensional space, effectively reducing its dimensions while preserving significant variance characteristics.

3. `main()`:

The `main` function is the entry point of the script and orchestrates the overall process. It handles command-line arguments to obtain the input data file, labels file, and output file name. It loads the dataset `X` and labels `y` from the specified files. It calls `calculate\_mixture\_scatter(X)` to compute the scatter matrix and then calls `reduce\_dimensions(X, scatter\_matrix)` to perform dimensionality reduction. Finally, it saves the reduced two-dimensional data to the specified output file and includes error handling to manage any exceptions that may occur during execution.

Scatter3.py –

The Python script performs dimensionality reduction on a dataset using the within-class scatter matrix. It takes three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced data will be saved. The script calculates the within-class scatter matrix, reduces the dimensions of the data to two using the eigenvalues and eigenvectors of the scatter matrix, and then saves the reduced data to the specified output file.

Function Summaries

`calculate\_within\_class\_scatter(data, labels)`

This function computes the within-class scatter matrix for a given dataset. It takes two parameters: `data`, which is a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function iterates over each unique class label, calculates the mean of the samples belonging to that class, and then computes the scatter of these samples around their mean. The resulting scatter matrices for each class are summed to produce the overall within-class scatter matrix, which quantifies the spread of samples within each class.

`reduce\_dimensions(data, scatter\_matrix)`

The `reduce\_dimensions` function performs dimensionality reduction on the input data using the provided scatter matrix. It first computes the eigenvalues and eigenvectors of the scatter matrix using NumPy's `linalg.eigh` function. The function then identifies the indices of the two smallest eigenvalues. A projection matrix is formed using the corresponding eigenvectors, and the original data is projected onto this lower-dimensional space, resulting in a reduced dataset that retains the most significant features.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function then calls `calculate\_within\_class\_scatter` to compute the scatter matrix and `reduce\_dimensions` to perform the dimensionality reduction. Finally, it saves the reduced data to the output file and handles any exceptions that may occur during file operations or calculations, providing appropriate error messages to the user.

Scatter4.py –

The Python script performs dimensionality reduction on a dataset using the within-class scatter matrix. It accepts three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced data will be saved. The script calculates the within-class scatter matrix, reduces the dimensions of the data to two using the eigenvalues and eigenvectors of the scatter matrix, and saves the reduced data to the specified output file. The script is structured to handle errors gracefully and provides feedback on the success or failure of the operations.

Function Summaries

`calculate\_within\_class\_scatter(data, labels)`

The `calculate\_within\_class\_scatter` function computes the within-class scatter matrix for a given dataset. It takes two parameters: `data`, a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function initializes a zero matrix for the scatter and iterates over each unique class label to calculate the mean of the samples belonging to that class. It then centers the samples by subtracting the class mean and computes the scatter for these centered samples. The resulting scatter matrices for each class are summed to produce the overall within-class scatter matrix, which quantifies the spread of samples within each class.

`reduce\_dimensions(data, scatter\_matrix)`

The `reduce\_dimensions` function performs dimensionality reduction on the input data using the provided scatter matrix. It computes the eigenvalues and eigenvectors of the scatter matrix using NumPy's `linalg.eigh` function. The function identifies the indices of the two largest eigenvalues. A projection matrix is formed using the corresponding eigenvectors, and the original data is projected onto this lower-dimensional space, resulting in a reduced dataset that retains the most significant features.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function calls `calculate\_within\_class\_scatter` to compute the scatter matrix and `reduce\_dimensions` to perform the dimensionality reduction. Finally, it saves the reduced data to the output file and handles any exceptions that may occur during file operations or calculations, providing appropriate error messages to the user. The function also prints a success message upon successful completion of the data reduction process.

Scatter5.py –

The Python script performs dimensionality reduction on a dataset using the between-class scatter matrix. It takes three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced data will be saved. The script calculates the between-class scatter matrix, reduces the dimensions of the data to two using the eigenvalues and eigenvectors of the scatter matrix, and saves the reduced data to the specified output file. The script includes error handling to manage issues related to file operations and calculations, providing feedback to the user.

Function Summaries

`calculate\_between\_class\_scatter(data, labels)`

The `calculate\_between\_class\_scatter` function computes the between-class scatter matrix for a given dataset. It takes two parameters: `data`, a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function first calculates the global mean of the data. It then initializes a zero matrix for the scatter and iterates over each unique class label to calculate the mean of the samples belonging to that class. The difference between the class mean and the global mean is computed, and the scatter contribution from each class is added to the overall scatter matrix, weighted by the number of samples in that class. The resulting matrix quantifies the spread of the class means relative to the overall mean.

`reduce\_dimensions(data, scatter\_matrix)`

The `reduce\_dimensions` function performs dimensionality reduction on the input data using the provided scatter matrix. It computes the eigenvalues and eigenvectors of the scatter matrix using NumPy's `linalg.eigh` function. The function identifies the indices of the two smallest eigenvalues. A projection matrix is formed using the corresponding eigenvectors, and the original data is projected onto this lower-dimensional space, resulting in a reduced dataset that retains the most significant features.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function calls `calculate\_between\_class\_scatter` to compute the scatter matrix and `reduce\_dimensions` to perform the dimensionality reduction. Finally, it saves the reduced data to the output file and handles any exceptions that may occur during file operations or calculations, providing appropriate error messages to the user. Upon successful completion, it prints a message indicating that the data has been successfully reduced and saved.

Scatter6.py –

The Python script performs dimensionality reduction on a dataset using the between-class scatter matrix. It takes three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced data will be saved. The script calculates the between-class scatter matrix, reduces the dimensions of the data to two using the eigenvalues and eigenvectors of the scatter matrix, and saves the reduced data to the specified output file. The script includes error handling to manage issues related to file operations and calculations, providing feedback to the user.

Function Summaries

`calculate\_between\_class\_scatter(data, labels)`

The `calculate\_between\_class\_scatter` function computes the between-class scatter matrix for a given dataset. It takes two parameters: `data`, a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function first calculates the global mean of the data across all samples. It initializes a zero matrix for the scatter and iterates over each unique class label to calculate the mean of the samples belonging to that class. For each class, it computes the difference between the class mean and the global mean, reshapes it for matrix multiplication, and updates the scatter matrix by adding the outer product of this mean difference, scaled by the number of samples in that class. The resulting matrix quantifies the spread of the class means relative to the overall mean.

`reduce\_dimensions(data, scatter\_matrix)`

The `reduce\_dimensions` function performs dimensionality reduction on the input data using the provided scatter matrix. It computes the eigenvalues and eigenvectors of the scatter matrix using NumPy's `linalg.eigh` function. The function identifies the indices of the two largest eigenvalues. A projection matrix is formed using the corresponding eigenvectors, and the original data is projected onto this lower-dimensional space, resulting in a reduced dataset that retains the most significant features.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function calls `calculate\_between\_class\_scatter` to compute the scatter matrix and `reduce\_dimensions` to perform the dimensionality reduction. Finally, it saves the reduced data to the output file and handles any exceptions that may occur during file operations or calculations, providing appropriate error messages to the user. Upon successful completion, it prints a message indicating that the data has been successfully reduced and saved.

Scatter7.py –

The Python script performs dimensionality reduction on a dataset using both the within-class and between-class scatter matrices. It takes three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced data will be saved. The script calculates the within-class and between-class scatter matrices, reduces the dimensions of the data to two using the eigenvalues and eigenvectors of these matrices, and saves the reduced data to the specified output file. The script includes error handling to manage issues related to file operations and calculations, providing feedback to the user.

Function Summaries

`calculate\_within\_class\_scatter(data, labels)`

The `calculate\_within\_class\_scatter` function computes the within-class scatter matrix for a given dataset. It takes two parameters: `data`, a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function initializes a zero matrix for the scatter and iterates over each unique class label to calculate the mean of the samples belonging to that class. It centers the samples by subtracting the class mean and computes the scatter for these centered samples using the dot product. The resulting scatter matrices for each class are summed to produce the overall within-class scatter matrix, which quantifies the spread of samples within each class.

`calculate\_between\_class\_scatter(data, labels)`

The `calculate\_between\_class\_scatter` function computes the between-class scatter matrix for a given dataset. It takes two parameters: `data`, a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function first calculates the global mean of the data across all samples. It initializes a zero matrix for the scatter and iterates over each unique class label to calculate the mean of the samples belonging to that class. For each class, it computes the difference between the class mean and the global mean, reshapes it for matrix multiplication, and updates the scatter matrix by adding the outer product of this mean difference, scaled by the number of samples in that class. The resulting matrix quantifies the spread of the class means relative to the overall mean.

`reduce\_dimensions(data, between\_scatter\_matrix, within\_scatter\_matrix)`

The `reduce\_dimensions` function performs dimensionality reduction on the input data using the provided between-class and within-class scatter matrices. It computes the eigenvalues and eigenvectors of the scatter matrices using NumPy's `eigh` function. The function identifies the indices of the two smallest eigenvalues. A projection matrix is formed using the corresponding eigenvectors, and the original data is projected onto this lower-dimensional space, resulting in a reduced dataset that retains the most significant features. The function also includes error handling to manage potential linear algebra errors during the computation.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function calls `calculate\_between\_class\_scatter` and `calculate\_within\_class\_scatter` to compute the respective scatter matrices and then calls `reduce\_dimensions` to perform the dimensionality reduction. Finally, it saves the reduced data to the output file and handles any exceptions that may occur during file operations or calculations, providing appropriate error messages to the user. Upon successful completion, it prints a message indicating that the data has been successfully reduced and saved.

Scatter8.py –

The Python script performs dimensionality reduction on a dataset using both the within-class and between-class scatter matrices. It takes three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced data will be saved. The script calculates the within-class and between-class scatter matrices, reduces the dimensions of the data to two using the eigenvalues and eigenvectors of these matrices, and saves the reduced data to the specified output file. The script includes error handling to manage issues related to file operations and calculations, providing feedback to the user.

Function Summaries

`calculate\_within\_class\_scatter(data, labels)`

The `calculate\_within\_class\_scatter` function computes the within-class scatter matrix for a given dataset. It takes two parameters: `data`, a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function initializes a zero matrix for the scatter and iterates over each unique class label to calculate the mean of the samples belonging to that class. It centers the samples by subtracting the class mean and computes the scatter for these centered samples using the dot product. The resulting scatter matrices for each class are summed to produce the overall within-class scatter matrix, which quantifies the spread of samples within each class.

`calculate\_between\_class\_scatter(data, labels)`

The `calculate\_between\_class\_scatter` function computes the between-class scatter matrix for a given dataset. It takes two parameters: `data`, a 2D NumPy array of feature values, and `labels`, a 1D array of class labels corresponding to each sample in the dataset. The function first calculates the global mean of the data across all samples. It initializes a zero matrix for the scatter and iterates over each unique class label to calculate the mean of the samples belonging to that class. For each class, it computes the difference between the class mean and the global mean, reshapes it for matrix multiplication, and updates the scatter matrix by adding the outer product of this mean difference, scaled by the number of samples in that class. The resulting matrix quantifies the spread of the class means relative to the overall mean.

`reduce\_dimensions(data, between\_scatter\_matrix, within\_scatter\_matrix)`

The `reduce\_dimensions` function performs dimensionality reduction on the input data using the provided between-class and within-class scatter matrices. It computes the eigenvalues and eigenvectors of the scatter matrices using NumPy's `eigh` function. The function identifies the indices of the two largest eigenvalues. A projection matrix is formed using the corresponding eigenvectors, and the original data is projected onto this lower-dimensional space, resulting in a reduced dataset that retains the most significant features. The function also includes error handling to manage potential linear algebra errors during the computation.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function calls `calculate\_between\_class\_scatter` and `calculate\_within\_class\_scatter` to compute the respective scatter matrices and then calls `reduce\_dimensions` to perform the dimensionality reduction. Finally, it saves the reduced data to the output file and handles any exceptions that may occur during file operations or calculations, providing appropriate error messages to the user. Upon successful completion, it prints a message indicating that the data has been successfully reduced and saved.

Selection1.py –

This Python script implements feature selection based on the Pearson correlation coefficient. It takes three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced dataset will be saved. The script calculates the absolute Pearson correlation between each feature and the labels, selects the two features with the highest correlations, and saves the reduced dataset containing only these features to the specified output file. The script includes error handling to manage issues related to file operations and calculations, providing feedback to the user.

Function Summaries

`pearson\_feature\_selection(X, y)`

The `pearson\_feature\_selection` function selects the two features that have the highest absolute Pearson correlation with the provided labels. It takes two parameters: `X`, a 2D NumPy array of feature values, and `y`, a 1D array of class labels. The function iterates over each feature in `X`, calculating the Pearson correlation coefficient between the feature and the labels using the `pearsonr` function from the `scipy.stats` module. It stores the absolute correlation values along with their corresponding feature indices in a list. After calculating the correlations for all features, the list is sorted in descending order based on the absolute correlation values. The function then selects the indices of the two features with the highest correlations and returns them as a list.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function calls `pearson\_feature\_selection` to determine the best features based on their Pearson correlation with the labels. It then reduces the dataset to include only the selected features and saves this reduced dataset to the output file. The function includes error handling to manage potential issues during file reading, feature selection, and data saving, providing appropriate error messages to the user. Upon successful completion, it prints a message indicating that the data has been successfully reduced and saved.

Selection2.py –

This Python script implements feature selection using the Fisher criterion for multiclass classification. It takes three command-line arguments: a data file containing feature values, a labels file indicating class labels, and an output file where the reduced dataset will be saved. The script calculates the Fisher scores for each feature, selects the two features with the highest scores, and saves the reduced dataset containing only these features to the specified output file. The script includes error handling to manage issues related to file operations and calculations, providing feedback to the user.

Function Summaries

`fisher\_criterion\_multiclass(feature, labels)`

The `fisher\_criterion\_multiclass` function computes the Fisher criterion for a single feature across multiple classes. It takes two parameters: `feature`, a 1D array of feature values, and `labels`, a 1D array of class labels corresponding to each sample. The function first identifies the unique classes and calculates the mean and variance of the feature values for each class. It then computes the between-class variance by summing the squared differences between the class means and the within-class variance by summing the variances of each class. The Fisher criterion is calculated as the ratio of the between-class variance to the within-class variance, providing a measure of how well the feature separates the classes. If there are fewer than two classes or if the within-class variance is zero, the function returns zero.

`fisher\_feature\_selection(X, y)`

The `fisher\_feature\_selection` function selects the best features based on the Fisher criterion. It takes two parameters: `X`, a 2D NumPy array of feature values, and `y`, a 1D array of class labels. The function iterates over each feature in `X`, calculating the Fisher score using the `fisher\_criterion\_multiclass` function. It stores the scores along with their corresponding feature indices in a list. After calculating the scores for all features, the list is sorted in descending order based on the scores. The function selects the indices of the two features with the highest scores and returns them as a list, indicating which features are most informative for classification.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. The function calls `fisher\_feature\_selection` to determine the best features based on the Fisher criterion. It then reduces the dataset to include only the selected features and saves this reduced dataset to the output file. The function includes error handling to manage potential issues during file reading, feature selection, and data saving, providing appropriate error messages to the user. Upon successful completion, it prints a message indicating that the data has been successfully reduced and saved.

Selection3.py –

This python script implements a feature selection process using Recursive Feature Elimination (RFE) with Linear Discriminant Analysis (LDA) as the estimator. It reads a dataset and corresponding labels from specified files, applies RFE to reduce the dataset to the two most significant features, and saves the reduced dataset to an output file. The script is designed to be run from the command line, requiring three arguments: the data file, the labels file, and the output file.

Function Summaries

`feature\_selection\_rfe(X, y)`

The `feature\_selection\_rfe` function takes two parameters: `X`, which is the feature matrix, and `y`, the target labels. It initializes a Linear Discriminant Analysis (LDA) model and applies Recursive Feature Elimination (RFE) to select the two most important features from the dataset. The function returns a reduced feature matrix containing only the selected features, allowing for a more focused analysis on the most relevant data.

`main()`

The `main` function serves as the entry point for the script. It checks for the correct number of command-line arguments and reads the data and labels from the specified files. If the files are successfully loaded, it calls the `feature\_selection\_rfe` function to perform feature selection and then saves the reduced dataset to the specified output file. The function also handles exceptions, providing error messages if any issues arise during file operations or processing.