Offline 04: PCA & EM

PCA

Idea/Intuition

Principal Component Analysis (PCA) is used for dimensionality reduction while preserving as much of the variability in the data as possible. It's often used in data preprocessing for machine learning and data visualization.

The core idea of PCA is to identify the directions (called principal components) in which the data varies the most. In other words, it finds the lines and planes that best summarize the data points.

Imagine you have a scatter plot of data in two dimensions. These data points might spread out more along one particular line or axis in space. PCA finds this line (the first principal component) and then finds the next line (second principal component) that captures the most variance or spread in the data that hasn't already been captured by the first principal component. These components are orthogonal (at right angles) to each other.

Here's why the principal components are orthogonal:

1. **Mathematical Basis**: PCA involves finding the eigenvectors of the covariance matrix of the data. These eigenvectors are orthogonal by the properties of eigenvalue decomposition. This orthogonality is inherent to the linear algebra techniques used in PCA.
2. **Maximizing Variance**: The first principal component is the direction in which the data varies the most. Once this direction is determined, the second principal component is the direction that is orthogonal to the first and captures the maximum of the remaining variance. This process continues for subsequent components.
3. **Data Representation**: The orthogonal components represent different dimensions of the data's variance. They provide a way to reorient the data into a new coordinate system where the axes (principal components) are uncorrelated, meaning that the variance along one axis (component) does not influence the variance along another.

Steps

1. **Centering the Data:**
    - `D_centered = D - np.mean(D, axis=0)`: Centers the data by subtracting the mean of each feature from the dataset. This step is crucial as PCA is affected by the scale of the data.

2. **Applying Singular Value Decomposition:**
    - `U, S, V_t = np.linalg.svd(D_centered, full_matrices=False)`: Decomposes the centered dataset into its singular vectors (`U`) and singular values (`S`). `V_t` (transpose of `V`) contains the principal components. The `full_matrices=False` argument is used to produce the reduced form of the SVD, which is more efficient.
3. **Projecting the Data onto the Principal Axes:**
    - `D = D_centered @ V_t.T[:, :2]`: Projects the data onto the first two principal components. This is done by multiplying the centered data with the first two columns of `V_t.T` (which is equivalent to `V`). This results in a dataset `D` that has been reduced to two dimensions.

GMM using EM Algorithm

Intuition Behind GMM

1. **Mixture of Gaussians**: Imagine you have a dataset, but you suspect that it's made up of several overlapping data clusters. Each of these clusters can be thought of as coming from different Gaussian distributions (each with its own mean and variance). GMM tries to model the data as a blend of these Gaussian distributions.
2. **Unknown Subpopulations**: In many cases, we don't know which data points belong to which underlying subpopulation. GMM helps in modeling this uncertainty. It assigns a probability (responsibility) indicating the likelihood of each data point belonging to each of the Gaussian distributions.

EM Algorithm

1. **Initialization**: First, the algorithm initializes the parameters of the Gaussian distributions. This can be random or more sophisticated methods like k-means clustering.
2. **Expectation Step (E-step)**: Calculate the responsibilities. This step involves estimating the probabilities that each data point belongs to each of the Gaussian distributions, given the current parameters. These probabilities are known as responsibilities.
3. **Maximization Step (M-step)**: Update the parameters. This step updates the parameters (means, variances, and mixture weights) of each Gaussian distribution in the mixture based on the responsibilities calculated in the E-step.
4. **Repeat Until Convergence**: These two steps (E and M) are repeated iteratively until the parameters converge (i.e., there is minimal change in the parameters between iterations).

Parameters

1. **`weights`**:
   - In a GMM, weights (also known as mixing coefficients) represent the probabilities associated with each Gaussian component in the mixture model. If there are `K` components in the model, you will have `K` weights, one for each component.
   - Weights determine the contribution of each Gaussian component to the overall model. They reflect how much a component is represented in the data set. For example, a higher weight for a particular component means that more data points are likely to be generated from that component. The sum of all weights equals 1, adhering to the probability distribution rule.
   - Initialized as a random array of length K, with each element representing the initial weight (or mixing proportion) of one Gaussian component in the mixture. The weights are normalized to sum up to 1, as they represent probabilities.
2. **`Means`**:
   - Means are the central points of the Gaussian components in the model. Each component has a mean value that represents its center in the feature space.
   - The mean of each Gaussian component helps in determining its location in the multidimensional feature space. It is a key parameter in defining the shape and orientation of the component. In the context of clustering, the mean can be thought of as the "center" of each cluster.
   - Initially set as a random 2D array of shape (K, 2), where each row represents the mean of one Gaussian component. The comment suggests a future implementation using k-means++ for better initialization.
3. **`Covariances`**:
   - Covariance matrices describe the shape and orientation of each Gaussian component in the model. For each component, the covariance matrix provides information on the variance of each feature and the covariance between different features.
   - A diagonal covariance matrix implies that features are uncorrelated, and the variance along each feature axis is independent. Non-diagonal elements indicate correlations between features. The shape of the Gaussian can be spherical, elongated, or rotated, depending on the covariance matrix.
   - An array of identity matrices, each of shape (2, 2), corresponding to the covariance matrices of each Gaussian component. This assumes that each component initially has unit variance along each feature dimension and no covariance between features.
4. **`Responsibilities`**:
   - Responsibilities represent the probability that each data point belongs to each of the Gaussian components. This is a matrix where each element indicates the

probability (responsibility) of a data point being generated by a particular Gaussian component.
    - Responsibilities are used in the Expectation step of the EM algorithm to update the weights, means, and covariances of the components. They are a key part of the model's iterative process to fit the data. By assigning probabilities rather than hard classifications, GMM provides a measure of uncertainty in the clustering process.
    - A 2D array of zeros with shape (N, K), where N is the number of data points. This array will store the responsibility of each Gaussian component for each data point, i.e., the probability that a data point belongs to a particular component.

Expectation Step

- **Purpose**: Computes the "responsibilities" or the probability of each data point belonging to each Gaussian component.
- **Process**:
    - It iterates over each Gaussian component (`K`).
    - For each component, it calculates the probability density of all data points (`self.data`) given the current mean (`self.means[k]`) and covariance (`self.covariances[k]`) of that component. This is done using the `multivariate_normal.pdf` function.
    - It multiplies these probabilities by the weight of the component (`self.weights[k]`). The probability density function (pdf) of a Gaussian distribution gives the likelihood of observing a data point given the parameters (mean and covariance) of that Gaussian. However, since we are dealing with a mixture, we need to adjust this likelihood by the probability (weight) of the data point being generated by that particular component. Mathematically, this is done by multiplying the Gaussian pdf value by the component's weight.
    - The responsibilities matrix (`self.responsibilities`) is updated, where each column corresponds to a component and each row to a data point.
    - The responsibilities are normalized for each data point so that their sum across all components equals 1.

Maximization Step

- **Purpose**: Updates the parameters of the Gaussian components based on the computed responsibilities.
- **Process**:
    - It calculates the sum of responsibilities for each component (`Nk`) which acts as an effective number of data points for each component.

  - Updates the weights by dividing `Nk` by the total number of data points (`N`). The new weight for each component is the proportion of points effectively assigned to it, as given by `Nk`, divided by the total number of data points `N`.
  - Calculates the new means as a weighted average of the data points, weighted by the responsibilities.
  - Calculates the new covariance matrices for each component. Each covariance matrix is computed as a weighted average of the outer products of the data points (centered by the new mean).
  - Adds a regularization term (`self.epsilon * np.eye(D)`) to the covariance matrices to ensure numerical stability.

Computing log-likelihood

- **Purpose**: Calculates the log-likelihood of the data given the model, a measure of how well the model fits the data.
- **Process**:
  - For each component, it computes the product of the weight and the probability density of all data points.
  - The log of the sum of these products across all components is computed for each data point.
  - The log-likelihood of the model is the sum of these log values across all data points.

AIC & BIC

AIC and BIC are measures that balance model fit (log-likelihood) with model complexity (number of parameters). They help in model selection by penalizing overcomplex models.

Plotting

In the plot, an ellipse represents a Gaussian component. The ellipse's position, size, and orientation are determined by the Gaussian's mean (`position`) and covariance matrix (`covariance`).

The orientation (`angle`) and size (`width`, `height`) of the ellipse are calculated using Singular Value Decomposition (SVD) of the covariance matrix. The dimensions of the ellipse (representing one standard deviation) are scaled by a factor of `2 * np.sqrt(2.0 * s)`, where `s` are the singular values from the SVD. This scaling helps in visualizing the spread of each component.

- **Size of the Ellipse**: The size of the ellipse is usually set to represent a certain confidence interval of the Gaussian distribution. For example, plotting the ellipse at one standard deviation encompasses approximately 68% of the probability mass for a bivariate normal distribution. This is often scaled to represent two or three standard deviations to visualize a larger portion of the distribution.
- **Shape of the Ellipse**: The shape (whether the ellipse is circular, elongated, etc.) is determined by the relative values of the eigenvalues. If the eigenvalues are equal or nearly equal, the ellipse will be more circular. If one eigenvalue is much larger than the other, the ellipse will be elongated in the direction of the corresponding eigenvector.
- **Rotation of the Ellipse**: The rotation angle of the ellipse in the plot is determined by the eigenvectors of the covariance matrix. The angle is calculated from the ratio of the components of the eigenvector associated with the largest eigenvalue.

Coding the plot

- **Covariance Matrix Decomposition**:
    - `U, s, Vt = np.linalg.svd(covariance)`: This line performs the Singular Value Decomposition (SVD) on the covariance matrix. SVD decomposes the matrix into its eigenvalues (`s`) and eigenvectors (`U` and `Vt`).
- **Ellipse Orientation (Angle)**:
    - `angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))`: The orientation of the ellipse is determined by the eigenvectors of the covariance matrix. Here, `arctan2` is used to calculate the angle of rotation of the ellipse in degrees based on the first eigenvector.
- **Ellipse Size (Width and Height)**:
    - `width, height = 2 * np.sqrt(2.0 * s)`: The width and height of the ellipse are based on the square roots of the eigenvalues (`s`) of the covariance matrix. The `sqrt(2.0 * s)` factor implies that the ellipse represents a contour of the Gaussian distribution corresponding to a specific confidence interval (like two standard deviations from the mean if `s` represents variances).