Spectral Methods

Spectral methods are a family of techniques used in machine learning for analyzing the properties of data through the spectrum of a matrix associated with the data. These methods are based on linear algebra and eigenvalue/eigenvector computations. They are widely used in data clustering, classification, denoising, signal separation, dimension reduction, etc. This introduction will cover several widely used spectral methods and related techniques to them.

PCA(Principal Component Analysis)

PCA is a widely used spectral method for dimensionality reduction in machine learning. It identifies and represents essential patterns and structures within high-dimensional data by transforming sets of correlated variables into uncorrelated principal components that capture significant data variation. There are multiple definitions of PCA that give rise to the same algorithm. There are various ways to formulate the PCA algorithm, all leading to the same results. In this discussion, I use the widely used **Maximum variance formulation** of PCA described in Chapter 12 of PRML. However, there are other interpretations of PCA, such as **Minimum-error formulation** and **Maximum likelihood formulation**, which you can refer to in the original book.

Assuming we have a dataset $\{x_i\}$ with dimensionality D where $i = 1, \dots, N$. We want to project the data onto space having dimensional $M < D$, while maximizing the variance of the projected data in order to retain the most important features of the dataset.

We can start by considering projection onto a one-dimensional space whose direction is given by u_1. This is to solve the following optimization problem

$$\begin{gathered} \max_{u_1} \ u_1^T S u_1 \\ \text{s.t. } u_1^T u_1 = 1 \end{gathered} \tag{34}$$

where S is the data covariance matrix defined by:

$$\begin{gathered} S = \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{x})^T \\ \bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n \end{gathered} \tag{35}$$

Note that the normalization constraint for u_1 is to prevent $\|s\| \to \infty$. This contained optimization problem can be directly solved by using a Lagrange multiplier denoted by λ_1:

$$\max_{u_1} \ u_1^T S u_1 + \lambda_1 (1 - u_1^T u_1) \tag{36}$$

whose solution is

$$S u_1 = \lambda_1 u_1 \tag{37}$$

Therefore we see that the u_1 is an eigenvector of S. If we insert this back to the criterion in Equation (1), we can obtain $u_1^T S u_1 = \lambda_1$. By analyzing this equation, it can be concluded that the maximum eigenvalue corresponds to the optimal solution. **We call the leading eigenvector of S the first principal axis of data samples and call the projection onto this axis $y_1 = u_1^T x$ the first principal component of the data sample.**

Next, we need to consider the next principal component which satisfies: $y_2 = u_2^T x$ and $Cov(y_1, y_2) = 0$. The second requirement is to decorrelate data samples in the projection space. This gives us an additional constrain:

$$Cov(y_1, y_2) = u_1^T S u_2 = \lambda_1 u_1^T u_2 = 0 \tag{38}$$

We can then write the optimization problem for u_2:

$$\begin{gathered}\max_{u_2}\ u_2^T S u_2 \\ \text{s.t. } u_2^T u_2 = 1,\ u_1^T u_2 = 0\end{gathered} \tag{39}$$

Solving this by introducing Lagrange multiplier denoted by λ_2 we obtain again:

$$S u_2 = \lambda_2 u_2 \tag{40}$$

To meet the given constraints, the u_2 should be the eigenvector of S corresponding to the second largest eigenvalue. We can repeat this process to obtain eigenvectors for all principal components (M of them). **We conclude that the principal vectors of a given dataset are the decreasing order eigenvectors of the covariance matrix, and the variance of the i'th principal component $y_i = u_i^T x$ is the eigenvalue** λ_i.

ICA (Independent Component Analysis)

Principal Component Analysis (PCA) is a method that enables the decorrelation of different dimensions of data samples by projecting them onto a new space where the covariances of dimensions are zero. This technique is useful in Blind Source Separation where the aim is to decompose input mixture signals into uncorrelated components. However, it is important to note that uncorrelatedness is a weaker form of independence. To address this limitation, **we can assume that the underlying source signals are independently distributed. This assumption leads to the Independent Component Analysis (ICA) method.**

Assuming that each of the observed data samples x is a mixture of M independent sources, we can represent it as the following equation:

$$x = As \tag{41}$$

where A is a $D \times M$ matrix containing the components and s is a vector of dimension M, containing the independent sources. Each dimension in x is a linear combination of different sources. In ICA, the goal is to recover the independent sources by estimating the unmixing matrix W. The unmixing matrix W is the inverse of the mixing matrix A, and it maps the mixed signals x to the independent sources s,:

$$s = Wx \tag{42}$$

The primary aim of ICA is to find the unmixing matrix W such that the components of s are statistically as independent as possible. To achieve this, we need to maximize the independence of different sources. **One approach to achieving this is to minimize the mutual information or to maximize the non-Gaussianity of s**. One popular method which adopts the latter approach is the FastICA algorithm. The intuition behind this approach straightforward: according to the Central Limit Theorem, the distribution of a sum of independent random variables tends towards a Gaussian distribution under certain conditions. Hence, a Gaussian signal can be considered as a linear combination of many independent signals. T**herefore, the separation of independent signals from their mixtures can be achieved by transforming the signals in such a way that they are as non-Gaussian as possible.** A more rigorous justification of the relationship between non-Gaussianity and independence is presented in the original paper by using mutual information.

To estimate the non-Gaussianity of the independent sources s in FASTICA, negentropy is used, which is based on differential entropy. The differential entropy H of a random vector y with density $f(y)$ is defined as

$$H(\mathbf{y})=-\int f(\mathbf{y}) \log f(\mathbf{y}) \mathrm{d} \mathbf{y} \tag{43}$$

An important result of this quantity is that a gaussian variable has the largest entropy among all random variables of equal variance, which implies that we can exploit this property as a measure of non-Gaussianity. This is usually done by using Negentropy J:

$$J(\mathbf{y})=H\left(\mathbf{y}_{\text {gauss }}\right)-H(\mathbf{y}) \tag{44}$$

where $\mathbf{y}_{\text {gauss }}$ is a Gaussian random variable of the same covariance matrix as y. However, estimating J directly in this way is intractable because it necessitates estimating the probability density function of y. To calculate this quantity, FASTICA uses the following approximation:

$$J(y) \propto[E\{G(y)\}-E\{G(v)\}]^2 \tag{45}$$

Therefore, by maximizing this expression, we can solve the unmixing problem and obtain independent signals. **In summary, ICA seeks to isolate statistically independent signals from their mixtures by maximizing the non-Gaussianity of each separated signal. In FASTICA, non-Gaussianity is measured using an approximation of Negentropy, and maximizing it can help us to extract independent signals.**

LDA(Linear Discriminant Analysis)

Both PCA and ICA are unsupervised techniques for discovering a projection space that meets specific goals, such as decorrelation of data dimensions or the recovery of independent sources. However, in certain cases, such as classification tasks, we work with labeled data and aim to project the data samples into spaces that can better differentiate different classes of data points. In such cases, Fisher's Linear Discriminant Analysis (FLDA or simply LDA) can be used. **LDA identifies a projection that maximizes the separation between different labeled classes while also minimizing the variance within the classes.** LDA provides a method for dimensionality reduction while retaining the class-discriminatory information, making it useful for supervised learning tasks.

Assuming the data samples $\{x_i\}$ are members of q classes $C_1, \ldots, C_q$. Let μ_j, S_j denote the center and covariance of class C_j , and μ denotes the center of the total data samples :

$$\begin{aligned} \mu_j &= \frac{1}{N_{C_j}} \sum_{x_i \in C_j} x_i \\ S_j &= \frac{1}{N_{C_j}} \sum_{x_i \in C_j} (x_n - \mu_j)(x_n - \mu_j)^T \\ \mu &= \frac{1}{N} \sum_{n=1}^{N} x_n \end{aligned} \tag{46}$$

The basic intuition behind Fisher's Linear Discriminant Analysis (LDA) is to find a projection direction that maximizes the separation between different classes' centers while reducing the variance of class distribution in the projected space. To evaluate the second requirement, we can consider the sum of the covariance matrix, defined as:

$$S_w = \sum_{j}^{q} S_j \tag{47}$$

Assuming the project direction is u, the "within class" variance in the projection direction can be expressed as $u^T S_w u$. To address the first requirement, we consider the variance of the class centers:

$$\begin{aligned} S_b &= \frac{1}{q} B B^T \\ B &= [\mu_1 - \mu, \ldots, \mu_q - \mu] \end{aligned} \tag{48}$$

The "between class" variance is then represented by $u^T S_b u$. By combining these results, we aim to maximize the objective function:

$$\max_u J(u) = \frac{u^T S_b u}{u^T S_w u} \tag{49}$$

This optimization problem is similar to the one encountered in the PCA section. The solution for this problem is given by:

$$S_b u = J(w) S_w u \tag{50}$$

Therefore, we see that the projection direction u are the eigenvectors of $S_w^{-1} S_b$ (assuming S_w is invertible).

In summary, Fisher's LDA finds the best projection direction to separate different classes while reducing class distribution variance. This relies on assessing "within class" and "between class" variances, combined into an objective function that is maximized. The eigenvectors of the matrix $S_w^{-1} S_b$ provide the optimal projection direction.

Metric Learning

Another supervised method involving the projection of data samples is metric learning. (There are also unsupervised metric learning methods, here I only focus on the supervised ones) The goal of this approach is to learn a distance metric from the training data that places data samples with the same class label closer together. Usually, Euclidean distance is used to measure the similarity between different data samples. However, this metric may not provide an accurate evaluation of similarity when the data structure is complicated. **In metric learning, a more flexible metric called Mahalanobis distances is used instead**:

$$D(x_i, x_j) = \sqrt{(x_i - x_j)^T M (x_i - x_j)} = \sqrt{(Lx_i - Lx_j)^T (Lx_i - Lx_j)} \tag{51}$$

where M is a semi-definite matrix and it can be decomposed as $M = L^T L$. The matrix L in metric learning defines a linear transformation of data samples, and the problem can be viewed as finding a feature space that can better evaluate the similarity between samples. If M is the identity matrix, , the traditional Euclidean metric is recovered. The main goal of metric learning is to determine optimal values for M or L based on the training data.

There are various metric learning methods, and in this introduction, I will only discuss the **Large Margin Nearest Neighbors (LMNN)** method, which is one of the most widely-used Mahalanobis distance learning methods. This method defines its constraint in a local way and focuses on neighbors. **The objective is to ensure that for every data point x_i, the k target neighbors (samples with the same labels) are close, and the impostors (samples with different labels) are far away**. To formulate this problem, let us first define the following notations:

$$\begin{aligned} \mathcal{S} &= \{(\boldsymbol{x}_i, \boldsymbol{x}_j) : l_i = l_j \text{ and } \boldsymbol{x}_j \text{ belongs to the } k\text{-neighborhood of } \boldsymbol{x}_i\} \\ \mathcal{R} &= \{(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k) : (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}, l_i \neq l_k\} \end{aligned} \tag{52}$$

The first term in the loss function should penalize large distances between each input and its target neighbors. In terms of the linear transformation L of the input space, the sum of these squared distances is given by:

$$\varepsilon_{\text{pull}}(\mathbf{L}) = \sum_{(x_i, x_j) \in \mathcal{S}} \|\mathbf{L}(x_i - x_j)\|^2 \tag{53}$$

The second term in the loss function penalizes small distances between differently labeled examples:

$$\varepsilon_{\text{push}}(\mathbf{L}) = \sum_{(x_i, x_j) \in \mathcal{S}} \sum_k (1 - y_{ik}) \left[1 + \left\|\mathbf{L}\left(\vec{x}_i - \vec{x}_j\right)\right\|^2 - \left\|\mathbf{L}\left(\vec{x}_i - \vec{x}_k\right)\right\|^2\right]_+ \tag{54}$$

where the term $[z]_+ = max(z, 0)$ denotes the standard hinge loss, $y_{ik} = \delta(l_i - l_k)$. The total loss function can be expressed as

$$\varepsilon(\mathbf{L}) = (1 - \mu)\varepsilon_{pull}(\mathbf{L}) + \mu\varepsilon_{push}(\mathbf{L}) \tag{55}$$

In the original paper, the authors reformulate this into a constrained convex optimization problem and develop an algorithm to solve it. Once trained, the learned transformation L can then be used for classification tasks or similarity evaluation.

Kernel PCA

In all the methods discussed above, linear transformations are applied to datasets to reveal the underlying data structures. However, it is often the case that a linear transformation is inadequate in capturing the full complexity of the structure. We can address this issue by using Kernel PCA, which is a non-linear version of PCA. **Kernel PCA finds the nonlinear data structure by mapping the data onto a higher-dimensional feature space and subsequently performing PCA on that space.** The transformation is expressed by an explicit function:

$$\Phi\left(\mathbf{x}_i\right) \text{ where } \Phi: \mathbb{R}^D \rightarrow \mathbb{R}^N \tag{56}$$

Working in the high-dimension space is very difficult. **To circumvent this issue, we can define a kernel function that calculates the dot production of high-dimension vectors:**

$$k(x_n, x_m) = \phi(x_n)^T \phi(x_m) \tag{57}$$

Later, this can be applied to simplify the calculation. To perform PCA, the first step is to express the covariance matrix in the N dimensional space, assuming zero-mean in this space:

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \phi\left(\mathbf{x}_n\right) \phi\left(\mathbf{x}_n\right)^{\mathrm{T}} \tag{58}$$

The principle axis is defined by this eigenvector expansion:

$$S u_i = \lambda_i u_i \tag{59}$$

$i = 1, .., M$. From this equation, it can be seen that u_i is expressed by a linear combination of $\phi(x_n)$:

$$\begin{gathered} \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\phi}\left(\mathbf{x}_n\right)\left\{\boldsymbol{\phi}\left(\mathbf{x}_n\right)^{\mathrm{T}} \mathbf{u}_i\right\} = \lambda_i \mathbf{u}_i \\ \mathbf{u}_i = \sum_{n=1}^{N} a_{in} \boldsymbol{\phi}\left(\mathbf{x}_n\right) \end{gathered} \tag{60}$$

Substituting the linear expression back into the eigenvector equation gives:

$$\frac{1}{N} \sum_{n=1}^{N} \phi\left(\mathbf{x}_n\right) \phi\left(\mathbf{x}_n\right)^{\mathrm{T}} \sum_{m=1}^{N} a_{im} \boldsymbol{\phi}\left(\mathbf{x}_m\right) = \lambda_i \sum_{n=1}^{N} a_{in} \boldsymbol{\phi}\left(\mathbf{x}_n\right). \tag{61}$$

This can be expressed in terms of the kernel function, by multiplying both sides by $\phi(x_l)^T$:

$$\frac{1}{N} \sum_{n=1}^{N} k\left(\mathbf{x}_l, \mathbf{x}_n\right) \sum_{m=1}^{N} a_{im} k\left(\mathbf{x}_n, \mathbf{x}_m\right) = \lambda_i \sum_{n=1}^{N} a_{in} k\left(\mathbf{x}_l, \mathbf{x}_n\right) \tag{62}$$

Vary l from 1 to N and express this in matrix notation, we obtain:

$$\mathbf{K}^2 \mathbf{a}_i = \lambda_i N \mathbf{K} \mathbf{a}_i \tag{63}$$

Here, K is a $N \times N$ matrix containing all kernel values, $\mathbf{a}_i$ is an N-dimensional vector containing coefficients for the ith PCA axis. To find solutions for $\mathbf{a}_i$, we can solve the following eigenvalue problem:

$$\mathbf{K} \mathbf{a}_i = \lambda_i N \mathbf{a}_i \tag{64}$$

We then normalize the coefficients by requiring the eigenvectors in feature space to be normalized:

$$1 = \mathbf{u}_i^{\mathrm{T}} \mathbf{u}_i = \sum_{n=1}^{N} \sum_{m=1}^{N} a_{in} a_{im} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_m) = \mathbf{a}_i^{\mathrm{T}} \mathbf{K} \mathbf{a}_i = \lambda_i N \mathbf{a}_i^{\mathrm{T}} \mathbf{a}_i \tag{65}$$

Combining these two equations, we can solve the $\mathbf{a}_i$. The projection of data samples onto the principle axis is then given by:

$$y_i(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \mathbf{u}_i = \sum_{n=1}^{N} a_{in} \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) = \sum_{n=1}^{N} a_{in} k(\mathbf{x}, \mathbf{x}_n) \tag{66}$$

Using this projection, we can then study the distribution of data points in the first several principal components. Note here we assume zero mean in the feature space. The general case requires an additional step for centering the kernel matrix K.

In conclusion, we define a high-dimension feature space using a kernel function, which calculates the similarity between pairs of data points. The principle axis in the feature space can help us capture the nonlinear relationships in data samples.

Reference

1. *Bishop, Christopher M. Pattern Recognition and Machine Learning. New York :Springer, 2006.*
2. *Introduction to Machine Learning*
3. Hyvärinen et al. Independent Component Analysis: Algorithms and Applications. Neural networks 2000
4. Weinberger et al. Distance Metric Learning for Large Margin Nearest Neighbor Classification. JMLR 2009
5. Wikipedia and ChatGPT