Introduction to Algorithms for Data Mining and Machine Learning

Chapter 5: Logistic Regression, PCA, LDA and ICA

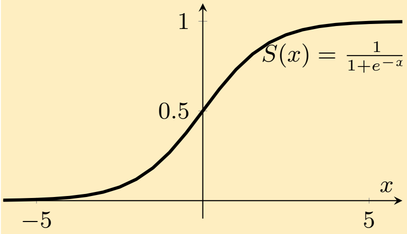
Xin-She Yang

For details, please read the book:

Xin-She Yang, **Introduction to Algorithms for Data Mining and Machine Learning**, Academic Press/Elsevier, (2019).

The dependence variable y_i is continuous in all the data regression and analysis we have done so far. If y_i is discrete or or simply binary with two values 1 (yes) and 0 (no), it becomes a classification problem and the regression becomes the logistic regression.

Logistic function and logit function



A logistic function,also called Sigmoid function, is defined as

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}, \quad x \in \mathbb{R},$$

which can be written as

$$S(x) = \frac{1}{2}\Big[1+\tanh(\frac{x}{2})\Big], \;\; \tanh(x) = \frac{e^x - x^{-x}}{e^x + e^{-x}}.$$

$S \to +1$ when $x \to +\infty$, while $S \to 0$ when $x \to -\infty$. In addition, $S \in (0, 1)$.

The derivative of $S(x)$ can be conveniently calculated by

$$S'(x) = \Big[\frac{1}{1+e^{-x}}\Big]' = \frac{-1}{(1+e^{-x})^2}(-e^{-x}) = \frac{1}{1+e^{-x}}\Big[1 - \frac{1}{(1+e^{-x})}\Big] = S(x)[1-S(x)].$$

The logit function is the inverse of $S(x)$ and can be defined as

$$\text{logit}(P) = \log\big(\frac{P}{1-P}\big) = \ln\big(\frac{P}{1-P}\big), \quad 0 < P < 1.$$

Logistic Regression

The simple logistic regression with one independent variable x and a binary dependent variable $y \in \{0,1\}$ with data points $(x_i, y_i)(i = 1, 2, ..., n)$ tries to fit a model of logistic probability

$$P = \frac{1}{1 + e^{a+bx}}.$$

This can be rewritten by using the logit function as

$$\ln\Big(\frac{P}{1-P}\Big) = a + bx,$$

which becomes a linear model in terms of the logit of probability P. In fact, the odds can be calculated from probability by

$$O_d(\text{odd}) = \frac{P}{1-P}, \quad P = \frac{O_d}{1+O_d},$$

which means that the logistic regression is a linear model of log(odds) versus x.
One naive way to solve this regression model is to convert it to nonlinear least squares

$$\text{Minimize} \quad \sum_{i=1}^{n} \Big[y_i - \frac{1}{1+e^{a+bx_i}}\Big]^2,$$

so as to solve a and b. However, this will not work in the true 'logistic-regression' sense.

For a given data set (x_i, y_i) with binary values $y_i \in \{0, \}$ $(i = 1, 2, ..., n)$, the proper binary logistic regression is to maximize the log-likelihood function. That is

$$\text{Maximize} \quad \log(L) = \sum_{i=1}^{n} \Big[y_i \ln P_i + (1 - y_i) \ln(1 - P_i) \Big], \qquad P_i = \frac{1}{1 + e^{a + bx_i}}. \tag{1}$$

This is based on the theory of the maximum likelihood probability. Since $y_i = 1$ or 0, the random variable Y for generating y_i obeys a Bernoulli distribution for probability P_i:

$$B_P(Y = y_i) = P_i^{y_i} (1 - P_i)^{1 - y_i},$$

so the joint probability of all data gives a likelihood function

$$L = \prod_{i=1}^{n} P(x_i)^{y_i} (1 - P(x_i))^{1 - y_i},$$

whose logarithm is given above in Eq. (1). As max L is the same as max $\log L$, the binary logistic regression is to fit the data such that the log-likelihood is maximized.

In principle, we can solve the above optimization problem in Eq. (1) by Newton's method or any other optimization techniques.

Let us use an example to explain the procedure in detail.

To fit a binary logistic regression using

$x : 0.1, \quad 0.5, \quad 1.0, \quad 1.5, \quad 2.0, \quad 2.5,$

$y : 0, \quad 0, \quad 1, \quad 1, \quad 1, \quad 0,$

we can use the following form

$$P_i = \frac{1}{1 + \exp(a + bx_i)}, \quad (i = 1, 2, ..., 6),$$

starting with initial values $a = 1$ and $b = 1$. Then, we have

$$P_i = \begin{pmatrix} 0.2497 & 0.1824 & 0.1192 & 0.0759 & 0.0474 & 0.0293 \end{pmatrix}.$$

The log-likelihood for each data point can be calculated by

$$L_i = y_i \ln P_i + (1 - y_i) \ln(1 - P_i),$$

and we have

$$L_i = [-0.2873, -0.2014, -2.1269, -2.5789, -3.0486, -0.0298], \quad \sum_{i=1}^{6} L_i = -8.2729.$$

Modifying a and b by Newton's method, then after about 20 iterations, we have

$$a = 0.8982, \quad b = -0.7099, \quad L_{\max} = -3.9162.$$

This means that the best-fit logistic regression model is

$$P = \frac{1}{1 + \exp(0.8982 - 0.7099x)}.$$

Exercise: `Implement this example in R or Python.`

The previous example has only one independent variable. In case of multiple independent variables $\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_m$, we can extend the above model as

$$y = \frac{1}{1 + \exp\left[w_0 + w_1\tilde{x}_1 + w_2\tilde{x}_2 + ... + w_m\tilde{x}_m\right]}.$$

Here, we use $\tilde{x}$ to highlight its variations. In order to write them compactly, let us define

$$\tilde{\boldsymbol{x}} = [1,\ \tilde{x}_1,\ \tilde{x}_2,\ ...,\tilde{x}_m]^T, \quad \boldsymbol{w} = [w_0,\ w_1,\ w_2,\ ...,w_m]^T,$$

where we have used 1 as a variable $\tilde{x}_0$ so as to eliminate the need to write w_0 everywhere. Thus, the logistic model becomes

$$P = \frac{1}{1 + \exp(\boldsymbol{w}^T\tilde{\boldsymbol{x}})},$$

which is equivalent to

$$\text{logit}(P) = \ln\frac{P}{1-P} = \boldsymbol{w}^T\tilde{\boldsymbol{x}}.$$

For all the data points $\tilde{\boldsymbol{x}}_i = [1, \tilde{x}_1^{(i)}, ..., \tilde{x}_m^{(i)}]$ with $y_i \in \{0, 1\}$ $(i = 1, 2, ..., n)$, we have

$$\text{maximize } \log(L) = \sum_{i=1}^{n}\Big[y_i \ln P_i + (1-y_i)\ln(1-P_i)\Big],$$

where $P_i = 1/[+\exp(\boldsymbol{w}^T\tilde{\boldsymbol{x}}_i)]$. The solution procedure is the same as before and can be obtained by any appropriate optimization algorithm.

Softmax Regression

Logistic regression is a binary classification technique with label $y_i \in \{0, 1\}$. For multi-class classification with $y_i \in \{1, 2, ..., K\}$ (with K different classes/labels), we can extend the logistic regression to the softmax regression, also called multinomial logistic regression.

Softmax

The softmax regression for probability $P(y = k|\tilde{\boldsymbol{x}})$ for $k = 1, 2, ..., K$ takes the form:

$$P(\tilde{\boldsymbol{x}}) = \frac{e^{\boldsymbol{w}^T \tilde{\boldsymbol{x}}}}{\sum_{j=1}^{K} e^{\boldsymbol{w}^T \tilde{\boldsymbol{x}}}},$$

where $\boldsymbol{w} = [w_0, w_1, ..., w_m]^T$ are the model parameters and w_0 is the bias. The m independent variables or attributes are written as a vector $\tilde{\boldsymbol{x}} = [1, x_1, ..., x_m]^T$ (with 1 for bias w_0). The denominator $\sum_1^K \exp[\boldsymbol{w}^T \tilde{\boldsymbol{x}}]$ is a normalization factor.

Similar to the log-likelihood in the logistic regression, for n data samples $(\tilde{\boldsymbol{x}}_i, y_i)$, the objective of softmax regression is to maximize the log-likelihood

$$\text{maximize } L = -\Big\{ \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{I}[y_i = k] \log \Big[\frac{e^{\boldsymbol{w}^T \tilde{\boldsymbol{x}}_i}}{\sum_{j=1}^{K} e^{\boldsymbol{w}^T \tilde{\boldsymbol{x}}_i}} \Big] \Big\},$$

where $\mathbb{I}[.]$ is an indicator function and $\mathbb{I}[y_i = k] = 1$ if $y_i = k$ is true (otherwise $\mathbb{I} = 0$ if $y_i \neq k$). However, there is no analytical solution for this optimization problem, but we can use a gradient-based optimizer (or any other suitable optimizer) to solve it.

PCA

The principal component analysis (PCA) is a dimensionality-reduction technique in representing data. For many quantities such as X_1, X_2, ..., X_p and y, it is desirable to represent the model as a hyperplane given by

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = \boldsymbol{X}^T \boldsymbol{\beta} + \beta_0.$$

In the simplest case, we have $y = \beta_0 + \beta_1 x$. For n data points (x_i, y_i), we can use the notations

$$\boldsymbol{x} = (x_1, x_2, ..., x_n), \quad \boldsymbol{y} = (y_1, y_2, ..., y_n),$$

so that we have

$$\boldsymbol{y} = \beta_0 + \beta_0 \boldsymbol{x}.$$

Now we can adjust the data so that their means are zero by subtracting their means $\bar{x}$ and $\bar{y}$, respectively. We have

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \bar{x}, \quad \tilde{\boldsymbol{y}} = \boldsymbol{y} - \bar{y},$$

and we have

$$\tilde{\boldsymbol{y}} + \bar{y} = \beta_0 + \beta_1 (\tilde{\boldsymbol{x}} + \bar{x}).$$

This leads to

$$\tilde{\boldsymbol{y}} = \beta_1 \tilde{\boldsymbol{x}},$$

where we have used $\bar{y} = \beta_0 + \beta_1 \bar{x}$. This is essentially equivalent to removing the bias β_0 (or zero bias) because the adjusted data have zero means.

The co-variance matrix can be written as

$$\boldsymbol{C} = \frac{1}{n-1} \begin{pmatrix} \tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^T & \tilde{\boldsymbol{x}}\tilde{\boldsymbol{y}}^T \\ \tilde{\boldsymbol{y}}\tilde{\boldsymbol{x}}^T & \tilde{\boldsymbol{y}}\tilde{\boldsymbol{y}}^T \end{pmatrix}.$$

Now we can extend this to the case of p variables. If the mean of each variable X_i is $\bar{X}_i (i = 1, 2, ..., p)$ with n data points, we now use $\tilde{\boldsymbol{X}} = (\tilde{X}_1 - \bar{X}_1, ..., \tilde{X}_p - \bar{X}_p)^T$ to denote the adjusted data with zero means (i.e., $\tilde{X}_i = X_i - \bar{X}_i$) in the following form

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} \tilde{\boldsymbol{X}}_1 & \tilde{\boldsymbol{X}}_2 & ... & \tilde{\boldsymbol{X}}_p \end{pmatrix}^T = \begin{pmatrix} \tilde{X}_1^{(1)} & \tilde{X}_1^{(2)} & ... & \tilde{X}_1^{(n)} \\ \tilde{X}_2^{(1)} & \tilde{X}_2^{(2)} & ... & \tilde{X}_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{X}_p^{(1)} & \tilde{X}_p^{(2)} & ... & \tilde{X}_p^{(n)} \end{pmatrix}, \quad \tilde{\boldsymbol{X}}_i = \boldsymbol{X}_i - \bar{X}_i.$$

Then, $\tilde{\boldsymbol{X}}$ is a $p \times n$ matrix where each column is a vector for a data point. Here we used a notation $\tilde{X}_i^{(j)}$ for the jth data observations for variable component i.

The co-variance matrix can be obtained by

$$\boldsymbol{C} = \frac{1}{n-1} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{X}}^T,$$

which is a $p \times p$ symmetric matrix. If the data values are all real numbers, then $\boldsymbol{C}$ is a real, symmetric matrix whose eigenvalues are all real, and the eigenvectors of two distinct eigenvalues are orthogonal to each other.

As the co-variance matrix $\boldsymbol{C}$ has a size of $p \times p$, it should in general have p eigenvalues $(\lambda_1, \lambda_2, ..., \lambda_p)$. Their corresponding (column) eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2$, ..., $\boldsymbol{u}_p$ should span an orthogonal matrix

$$\tilde{\boldsymbol{U}} = \begin{pmatrix} \vdots & \vdots & ... & \vdots \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & ... & \boldsymbol{u}_p \\ \vdots & \vdots & ... & \vdots \end{pmatrix} \in \mathbb{R}^{p \times p},$$

which has the orthogonal properties $\tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^T = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^T = \boldsymbol{I}$ (an identity matrix) and

$$\tilde{\boldsymbol{U}}^{-1} = \tilde{\boldsymbol{U}}^T.$$

The component associated with the principal direction (of the eigenvector) of the largest eigenvalue $\lambda* = \max\{\lambda_i\}$ is the first main component. This corresponds to the rotation of base vectors so as to align the main component to this principal direction, rotated by

$$\boldsymbol{Y} = \tilde{\boldsymbol{U}}^T \tilde{\boldsymbol{X}}.$$

The principal component analysis (PCA) essentially intends to transform a set of data points (for p variables that may be correlated) into a set of $k < p$ principal components which are a set of transformed points of linearly uncorrelated variables. It is essentially to identify the first (major) component with the most variations from the co-variance matrix, then identify the next component with second most variations. Other components can be identified in a similar manner.

Example

From the following data:

$$\begin{cases} x: & -1.0 & +0.0 & +1.0 & +2.0 & +3.0 & +4.0 \\ y: & +1.1 & +0.7 & +2.3 & +1.4 & +2.2 & +3.7 \end{cases}$$

we first adjust the data by subtracting their means $\bar{x}$ and $\bar{y}$, respectively,

$$\bar{x} = \frac{1}{6}\sum_{i=1}^{6} x_i = 1.5, \quad \bar{y} = \frac{1}{6}\sum_{i=1}^{6} y_i = 1.9.$$

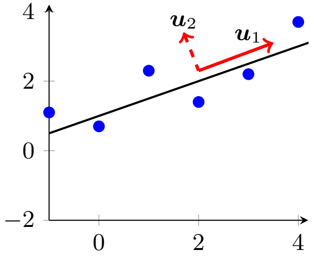
We have $X = x - \bar{x}$ and $Y = y - \bar{y}$

$$\begin{cases} X: & -2.5 & -1.5 & -0.5 & +0.5 & +1.5 & +2.5 \\ Y: & -0.8 & -1.2 & +0.4 & -0.5 & +0.3 & +1.8 \end{cases}$$

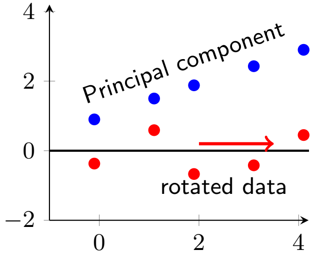
that have zero means. We can then calculate the co-variance matrix

$$\boldsymbol{C} = \begin{pmatrix} 3.500 & 1.660 \\ 1.660 & 1.164 \end{pmatrix}.$$

Since cov(x, y) is positive [so is cov(X, Y)=cov(x, y)], it is expected that y increases with x or vice versa.



$\Longrightarrow$



The two eigenvalues of $\boldsymbol{C}$ are $\lambda_1 = 4.36$ and $\lambda_2 = 0.302$. Their corresponding eigenvectors are $\boldsymbol{u}_1 = (0.887, 0.461)^T$ and $\boldsymbol{u}_2 = (-0.461, 0.887)^T$, respectively. They can span an orthogonal matrix

$$\tilde{\boldsymbol{U}} = \begin{pmatrix} 0.887 & -0.461 \\ 0461 & 0.887 \end{pmatrix}.$$

Using the adjusted data

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} -2.5 & -1.5 & -0.5 & +0.5 & +1.5 & +2.5 \\ -0.8 & -1.2 & +0.4 & -0.5 & +0.3 & +1.8 \end{pmatrix}$$

we have the rotated data $\boldsymbol{Y} = \tilde{\boldsymbol{U}}^T \tilde{\boldsymbol{X}}$, which gives [0.44, -0.37, 0.59, -0.67, -0.42, 0.45] that are shown (in red) in the above figure.

Since λ_1 the larger eigenvalue, its corresponding eigenvector indicates that the x direction is the main component (see the figure).

LDA

The main idea of linear discriminant analysis (LDA) is to find a proper projection or transformation maximizing the separability of different classes.

For n data points $(\boldsymbol{x}^{(i)}, y^{(i)})$ with $\boldsymbol{x}^{(i)} = (x_1^{(i)}, ..., x_m^{(i)})^T$, the output $y^{(i)}$ belongs to K different classes. In the simplest case, let us start with $K = 2$ where $y^{(i)}$ can belong to either class 1 (C_1) or class 2 (C_2) (for example, $y \in \{0, 1\}$ with $C_1 = 0$ and $C_2 = 1$). There are n_1 data points belong to C_1 and n_2 points belong to C_2 so that $n_1 + n_2 = n$.

For a two-class model with $y = \boldsymbol{w}^T \boldsymbol{x} = w_0 + x_1 x_1 + ... + w_m x_m$, we can define their means as

$$\boldsymbol{\mu}_1 = \frac{1}{n_1} \sum_{i \in C_1} \boldsymbol{x}^{(i)}, \quad \boldsymbol{\mu}_2 = \frac{1}{n_2} \sum_{i \in C_2} \boldsymbol{x}^{(i)},$$

which are vectors of m components. Their covariance matrices can be calculated by

$$\boldsymbol{S}_1 = \frac{1}{n_1 - 1} \sum_{i \in C_1} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_1)(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_1)^T,$$

$$\boldsymbol{S}_2 = \frac{1}{n_2 - 1} \sum_{i \in C_2} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_2)(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_2)^T.$$

Both are $m \times m$ matrices.

The main objective is to maximize the mean distance (or distance squared) between the two classes, which can be written as

$$\text{maximize} \;\; f(\boldsymbol{w}) = \frac{[\boldsymbol{w}^T(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)]^2}{\boldsymbol{w}^T \boldsymbol{S} \boldsymbol{w}},$$

where $\boldsymbol{S} = n_1 \boldsymbol{S}_1 + n_2 \boldsymbol{S}_2$. This is an optimization problem.

It can be shown that the optimal solution is

$$\boldsymbol{w} = \boldsymbol{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2).$$

In practice, we have to solve an eigenvalue problem to find the transformation or projection.

Let use use an example to show how it works.

Example

Suppose we have 12 data points where 6 belong to one class C_1 and 6 points belong to C_2, so we have $n_1 = n_2 = 6$ and $n = 12$. Their data points are

$$\boldsymbol{x}_{C_1} = \left| \begin{array}{cccccc} 1 & 2 & 2 & 3 & 1.6 & 3 \\ \hline 2 & 1 & 1.5 & 2 & 1.7 & 3 \end{array} \right|, \quad \boldsymbol{x}_{C_2} = \left| \begin{array}{cccccc} 5 & 6 & 7 & 8 & 9 & 7 \\ \hline 4 & 5 & 4 & 5.5 & 6.5 & 8 \end{array} \right|.$$

It is straightforward to calculate that

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 2.1 \\ 1.7 \end{pmatrix}, \quad \boldsymbol{\mu}_2 = \begin{pmatrix} 7 \\ 5.5 \end{pmatrix}.$$

The covariances are

$$\boldsymbol{S}_1 = \frac{1}{5}\begin{pmatrix} 3.1 & 2.3 \\ 2.3 & 2.8 \end{pmatrix} = \begin{pmatrix} 0.62 & 0.46 \\ 0.46 & 0.56 \end{pmatrix}, \quad \boldsymbol{S}_2 = \begin{pmatrix} 2 & 1.1 \\ 1.1 & 2.4 \end{pmatrix}.$$

Thus, we have

$$\boldsymbol{S} = n_1 \boldsymbol{S}_1 + n_2 \boldsymbol{S_2} = \begin{pmatrix} 13.1 & 7.8 \\ 7.8 & 14.8 \end{pmatrix},$$

and

$$\boldsymbol{w} = \boldsymbol{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = -\begin{pmatrix} 0.3223 \\ 0.0869 \end{pmatrix},$$

which can be normalized into a unit vector as

$$\boldsymbol{w}_n = \frac{\boldsymbol{w}}{||\boldsymbol{w}||} = \begin{pmatrix} -0.9655 \\ -0.2603 \end{pmatrix}.$$

This has the same direction as

$$\begin{pmatrix} 0.9655 \\ 0.2603 \end{pmatrix},$$

which is the direction onto which the data points should project so as to give the maximum separability of the two classes.

Both LDA and PCA can be very effective for many applications such as pattern recognition.

SVD

A real matrix $\boldsymbol{A}$ of size $m \times n$ can always be written as

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{Q}^T,$$

where $\boldsymbol{P}$ is an $m \times m$ orthonormal matrix, and $\boldsymbol{Q}$ is an $n \times n$ orthonormal matrix, while $\boldsymbol{D}$ is an $m \times n$ diagonal matrix. This is called the singular value decomposition (SVD).

A square matrix $\boldsymbol{Q}$ is orthonormal if it is orthogonal and each column is a unit vector. That is

$$\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}, \quad \boldsymbol{Q}^{-1} = \boldsymbol{Q}^T,$$

where $\boldsymbol{I}$ is an identity matrix of the same size as $\boldsymbol{Q}$. Similarly, we have

$$\boldsymbol{P}\boldsymbol{P}^T = \boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}, \quad \boldsymbol{P}^{-1} = \boldsymbol{P}^T.$$

In addition, the $m \times n$ diagonal matrix $\boldsymbol{D}$ is defined by $D_{ii} = d_i \geq 0$ and $D_{ij} = 0$ when $i \neq j$ for all $i = 1, ..., m$ and $j = 1, 2, ..., n$. For example, where $m = 3$ and $n = 4$, we have

$$\boldsymbol{D} = \begin{pmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ 0 & 0 & d_3 & 0 \end{pmatrix}.$$

There are rigorous algorithms to carry out SVD effectively and interested readers can refer to more specialized literature, starting with the wikipedia article.

PCA is mainly based on the second-order statistics such as variance. Data projection can be carried out using SVD.

If we can rewrite $\tilde{\boldsymbol{X}}$ using the following SVD:

$$\tilde{\boldsymbol{X}} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{Q}^T,$$

we have the covariance matrix

$$\boldsymbol{C} = \frac{1}{n-1}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T = \frac{1}{n-1}\big(\boldsymbol{P}\boldsymbol{D}\boldsymbol{Q}^T\big)\big(\boldsymbol{P}\boldsymbol{D}\boldsymbol{Q}^T\big)^T$$

$$= \frac{1}{n-1}\boldsymbol{P}\boldsymbol{D}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{D}^T\boldsymbol{P}^T = \frac{1}{n-1}\boldsymbol{P}\boldsymbol{D}^2\boldsymbol{P}^T = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^T,$$

where we have used $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$. Here, $\boldsymbol{\Lambda} = \boldsymbol{D}^2/(n-1)$ is a diagonal matrix where diagonal values are the eigenvalues of the eigenvectors (columns of $\boldsymbol{P}$).

Thus, the data projection in PCA is equivalent to

$$\boldsymbol{Y} = \tilde{\boldsymbol{U}}^T\tilde{\boldsymbol{X}} = (\tilde{\boldsymbol{U}}^T\boldsymbol{P})\boldsymbol{D}\boldsymbol{Q}^T.$$

This means that PCA can be viewed as a special class of SVD.

ICA

The directions obtained in PCA are orthogonal and PCA uses means and variances (up to the second-order). It would be advantageous to use high-order statistical information for some data sets, leading to the independent component analysis (ICA).

ICA

The main idea of ICA is that the n observations $x_i (i = 1, 2, ..., n)$ are a linear mixture of n independent components $s_j (j = 1, 2, ..., n)$. That is

$$x_i = a_{i1} s_1 + a_{i2} s_2 + ... + s_{in} s_n = \sum_{j=1}^{n} a_{ij} s_j.$$

These independent components can be considered as source signals or as hidden random variables, thus are called latent variables or hidden features. We can rewrite the above equation as

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{s},$$

where

$$\boldsymbol{x} = [x_1, x_2, ..., x_n]^T, \quad \boldsymbol{A} = [a_{ij}], \quad \boldsymbol{s} = [s_1, s_2, ..., s_n]^T.$$

The main task is to estimate both the mixing matrix $\boldsymbol{A}$ and the unknown components $\boldsymbol{s}$.

The main assumptions for $s_j (j = 1, 2, ..., n)$ are that they must be mutually statistically independent and must be non-Gaussian. Otherwise, the ICA will not work.

The idea is find an approximation $\boldsymbol{W}$ as an estimate to $\boldsymbol{A}^{-1}$ and $\boldsymbol{s} = \boldsymbol{W}\boldsymbol{x}$.
For any real symmetric matrix $\boldsymbol{C}$, we have its eigenvalue problem as $\boldsymbol{C}\boldsymbol{u} = \lambda\boldsymbol{u}$. We can form an orthogonal matrix $\boldsymbol{U}$ whose columns are the eigenvectors

$$\boldsymbol{C}\boldsymbol{U} = \boldsymbol{U}\boldsymbol{\Lambda},$$

where Λ is a diagonal matrix whose diagonal values are the eigenvalues λ_i of $\boldsymbol{C}$ with its corresponding eigenvector in ith column of $\boldsymbol{U}$. Thus, we have

$$\boldsymbol{C} = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^{-1} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T,$$

which is the well-known eigenvalue decomposition (EVD).

Whitening

We can scale $\boldsymbol{x}$ such that its variance becomes an identity matrix $\boldsymbol{I}$

$$\tilde{\boldsymbol{x}} = \big(\boldsymbol{U}\boldsymbol{D}^{-1/2}\boldsymbol{U}^T\big)\boldsymbol{x},$$

with $\boldsymbol{D}^{-1/2}$=diag$\{d_1^{-1/2}, ..., d_n^{-1/2}\}$, we have its covariance matrix

$$\tilde{\boldsymbol{C}} = E[\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^T] = \boldsymbol{I},$$

which is called whitening that is to normalize the variances after rotating the centered observations $\bar{x}_i$ to align with the eigenvectors of the covariance matrix to the rotated basis vectors. Thus, all linear dependencies in the data have been removed.

Many software packages have implemented both PCA and ICA.

References

References

- Xin-She Yang, **Introduction to Algorithms for Data Mining and Machine Learning**, Academic Press/Elsevier, (2019).
- Xin-She Yang, **Optimization Techniques and Applications with Examples**, John Wiley & Sons, (2018).

Notes on Software

Almost all major software packages in statistics, data mining and machine learning have implemented the algorithms we introduced here in this chapter.

- R has `pca` and `fastICA` packages.
- Python has `pca`, `ica` from `sklearn`.
- Matlab all have all the functionalities such as `pca`, `svd` and `mnrfit` for multinomial logistic regression.

Any questions? **Thank you :)**