1 Singular Value Decomposition and Principal Component Analysis

In these lectures we discuss the SVD and the PCA, two of the most widely used tools in machine learning. Principal Component Analysis (PCA) is a linear dimensionality reduction method dating back to Pearson (1901) and it is one of the most useful techniques in exploratory data analysis. It is also known under different names such as the Karhunen-Love Transform, the Hotelling transform, and Proper Orthogonal Decomposition (POD). PCA can be applied to a data set comprising of n vectors $x_1, \dots, x_n \in \mathbb{R}^d$ and in turn returns a new basis for $\mathbb{R}^d$ whose elements are terms the principal components. It is important that the method is completely data-dependent, that is, the new basis is only a function of the data. The PCA builds on the SVD (or the spectral theorem), we therefore start with the SVD.

1.1 Singular Value Decomposition (SVD)

Consider a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ or $\mathbb{C}^{m \times n}$ and let us assume that $m \geq n$. Then the *singular value decomposition* (SVD) of $\boldsymbol{A}$ is given by [1]

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{W}^*,$$

where $\boldsymbol{U}$ is $m \times m$, $\boldsymbol{D}$ is $m \times n$, $\boldsymbol{W}$ is $n \times n$, $\boldsymbol{U}$ and $\boldsymbol{W}$ are unitary (i.e., $\boldsymbol{U}^*\boldsymbol{U} = \boldsymbol{U}\boldsymbol{U}^* = \boldsymbol{I}_m$, $\boldsymbol{W}\boldsymbol{W}^* = \boldsymbol{W}^*\boldsymbol{W} = \boldsymbol{I}_n$), and $\boldsymbol{D}$ is a diagonal (rectangular) matrix

$$\boldsymbol{D} = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_n \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

with $D_{ii} = \sigma_i > 0$. Here, σ_i are called the *singular values* of $\boldsymbol{A}$, the columns of $\boldsymbol{U}$ are the corresponding *left singular vectors*, and the columns of $\boldsymbol{W}$ are the corresponding *right singular vectors*.

Let $\boldsymbol{U} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_m]$, $\boldsymbol{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w}_n]$ and let r be the rank of $\boldsymbol{A}$. Then we can write

$$\boldsymbol{A} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{w}_i^*,$$

with $r \leq n$ (and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$). (So $\boldsymbol{A}$ is a sum of weighted rank-one matrices.) The SVD exists for any finite-dimensional matrix.

Remarks:

- The $\boldsymbol{u}_i$ are eigenvectors of $\boldsymbol{A}\boldsymbol{A}^*$ and the $\boldsymbol{w}_i$ are eigenvectors of $\boldsymbol{A}^*\boldsymbol{A}$.
- $\boldsymbol{A}\boldsymbol{A}^*$ and $\boldsymbol{A}^*\boldsymbol{A}$ are positive semidefinite so their eigenvalues are nonnegative.
- If λ_i are the eigenvalues of $\boldsymbol{A}^*\boldsymbol{A}$, then $\sigma_i^2 = \lambda_i$ if $\lambda_i > 0$. (Here we're saying that singular values must be positive, but this is more of a matter of taste.)
- If $\boldsymbol{A}$ is square and Hermitian, then the SVD and the eigenvalue decomposition are the same.
- We could alternatively define the SVD with $\boldsymbol{U}$ as an $m \times n$ matrix, $\boldsymbol{D}$ as an $n \times n$ matrix, and $\boldsymbol{W}$ as an $n \times n$ matrix. In this case, $\boldsymbol{U}^*\boldsymbol{U} = \boldsymbol{I}_n$, and $\boldsymbol{W}^*\boldsymbol{W} = \boldsymbol{W}\boldsymbol{W}^* = \boldsymbol{I}_n$.

Some intuition for SVD: SVD rotates the matrix $\boldsymbol{A}$ by $\boldsymbol{U}$ and $\boldsymbol{W}^*$ so that $\boldsymbol{A}$ becomes a diagonal matrix.

2 Principal Component Analysis (PCA)

2.1 Motivation

Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, we want to project the $\boldsymbol{x}_i$ onto $\mathbb{R}^k$, $k < d$. So, how do we choose k and the orientation of the subspace? We consider two ideas:

1. Find the k-dimensional subspace for which the projections of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ best approximate the original points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. (We define "best approximation" in the sense of the 2-norm.)
2. We also want to conserve what makes the data points different from each other. Hence, find the k-dimensional projection of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ that preserves most of the variance of the $\boldsymbol{x}_i$.

Both of the two ideas above are solved by *principal component analysis* (PCA).

2.2 Optimization problem formulation [following lecture notes of Singer and Bandeira]

We denote the *sample mean* by

$$\boldsymbol{\mu}_n := \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$$

and *sample covariance* matrix by

$$\boldsymbol{\Sigma}_n := \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu}_n) (\boldsymbol{x}_i - \boldsymbol{\mu}_n)^* .$$

Let us focus on the first idea in Section 2.1. We want to approximate each $\boldsymbol{x}_i$ by an affine low-dimensional subspace such that for each $\boldsymbol{x}_i$ we have

$$\boldsymbol{x}_i \approx \boldsymbol{\mu} + \sum_{j=1}^{k} (\boldsymbol{\alpha}_i)_j \, \boldsymbol{v}_j,$$

where $\boldsymbol{V} := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]$ is an orthonormal basis to be determined. We can rewrite the above as

$$\boldsymbol{x}_i \approx \boldsymbol{\mu} + \boldsymbol{V}\boldsymbol{\alpha}_i,$$

where

$$\boldsymbol{\alpha}_i = \begin{bmatrix} \alpha_{i1} \\ \alpha_{i2} \\ \vdots \\ \alpha_{ik} \end{bmatrix},$$

with $\boldsymbol{V}$ as a $n \times k$ matrix satisfying $\boldsymbol{V}^*\boldsymbol{V} = \boldsymbol{I}_k$. Now, we try to solve the optimization problem

$$\min_{\substack{\boldsymbol{\mu},\ \boldsymbol{V},\ \boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_n \\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k}} I := \sum_{i=1}^{n} \| \boldsymbol{x}_i - (\boldsymbol{\mu} + \boldsymbol{V}\boldsymbol{\alpha}_i) \|_2^2.$$

Thus, we try to minimize the ℓ_2-error across all vectors $\boldsymbol{x}_i$. (Unlike in the JL approach we do not strive for minimizing the error uniformly (within an ε-range) across all $\boldsymbol{x}_i$, but rather the *average error.*)

2.3 Solving the optimization problem

Fortunately we can separate this problem and first optimize over μ, then $\boldsymbol{\alpha}$, then over $\boldsymbol{V}$. (There are optimization problems which look similar but where you can't do this strategy of separation of variables.)

Let us first optimize with respect to $\boldsymbol{\mu}$. Without loss of generality, we can assume that $\sum_{i=1}^{n} \boldsymbol{\alpha}_i = 0$, because otherwise we could absorb the nonzero $\sum_i \boldsymbol{\alpha}_i$ into $\boldsymbol{\mu}$. Then:

$$\frac{\partial I}{\partial \boldsymbol{\mu}} = -2 \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{V}\boldsymbol{\alpha}_i) .$$

Setting the right-hand side equal to zero, we get

$$\begin{aligned}\boldsymbol{\mu} &= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i \\ &= \boldsymbol{\mu}_n.\end{aligned}$$

Now let's optimize in α. We calculate:

$$\frac{\partial I}{\partial \boldsymbol{\alpha}_i} = \left(\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{V}\boldsymbol{\alpha}_i\right)^* \boldsymbol{V}.$$

Setting the right-hand side equal to zero, we get

$$\boldsymbol{\alpha}_i = \boldsymbol{V}^*\left(\boldsymbol{x}_i - \boldsymbol{\mu}\right).$$

Plugging in the expressions for μ and α_i into I, we get

$$I = \sum_{i=1}^{n}\left\|\boldsymbol{x}_i - \boldsymbol{\mu}_n - \boldsymbol{V}\boldsymbol{V}^*\left(\boldsymbol{x}_i - \boldsymbol{\mu}_n\right)\right\|_2^2$$

where $\boldsymbol{V}\boldsymbol{V}^*$ is an orthogonal projection matrix. Thus, letting $\boldsymbol{y}_i := \boldsymbol{x}_i - \boldsymbol{\mu}_n$,

$$I = \sum_{i=1}^{n}\|\boldsymbol{y}_i - \boldsymbol{V}\boldsymbol{V}^*\boldsymbol{y}_i\|_2^2.$$

Denote $\boldsymbol{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n]$. Then

$$\begin{aligned}\min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k}\sum_{i=1}^{n}\|\boldsymbol{y}_i - \boldsymbol{V}\boldsymbol{V}^*\boldsymbol{y}_i\|_2^2 &= \min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \operatorname{trace}\left[\left(\boldsymbol{Y} - \boldsymbol{V}\boldsymbol{V}^*\boldsymbol{Y}\right)^*\left(\boldsymbol{Y} - \boldsymbol{V}\boldsymbol{V}^*\boldsymbol{Y}\right)\right]. \\ &= \min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \operatorname{trace}\left[\boldsymbol{Y}^*(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*)(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*)\boldsymbol{Y}\right].\end{aligned}$$

Using properties of the trace (i.e., the *circular shift* property and linearity), and the fact that $(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*)(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*) = \boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*$,

$$\begin{aligned}\min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k}\sum_{i=1}^{n}\|\boldsymbol{y}_i - \boldsymbol{V}\boldsymbol{V}^*\boldsymbol{y}_i\|_2^2 &= \min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \operatorname{trace}\left[\boldsymbol{Y}\boldsymbol{Y}^*(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*)(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*)\right] \\ &= \min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \operatorname{trace}\left[\boldsymbol{Y}\boldsymbol{Y}^*(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^*)\right] \\ &= \min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \left[\operatorname{trace}\left(\boldsymbol{Y}\boldsymbol{Y}^*\right) - \operatorname{trace}\left(\boldsymbol{Y}\boldsymbol{Y}^*\boldsymbol{V}\boldsymbol{V}^*\right)\right] \\ &= \min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \left[\operatorname{trace}\left(\boldsymbol{Y}\boldsymbol{Y}^*\right) - \operatorname{trace}\left(\boldsymbol{Y}\boldsymbol{Y}^*\boldsymbol{V}\boldsymbol{V}^*\right)\right] \\ &= \min_{\boldsymbol{V}:\ \boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \left[\operatorname{trace}\left(\boldsymbol{Y}\boldsymbol{Y}^*\right) - \operatorname{trace}\left(\boldsymbol{V}^*\boldsymbol{Y}\boldsymbol{Y}^*\boldsymbol{V}\right)\right]. \qquad (1)\end{aligned}$$

But $\boldsymbol{Y}$ does not depend on $\boldsymbol{V}$! Hence, the minimum in (1) is idependent of the expression $\operatorname{trace}(\boldsymbol{Y}\boldsymbol{Y}^*)$ and thus coincides with the solution to

$$\max_{\boldsymbol{V}\,:\,\boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \frac{1}{n}\operatorname{trace}(\boldsymbol{V}^*\boldsymbol{Y}\boldsymbol{Y}^*\boldsymbol{V}) = \max_{\boldsymbol{V}\,:\,\boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \operatorname{trace}(\boldsymbol{V}^*\boldsymbol{\Sigma}_n\boldsymbol{V}).$$

Let $\boldsymbol{\Sigma}_n$ have eigenvalue decomposition

$$\boldsymbol{\Sigma}_n = \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^*,$$

where $\lambda_i \geq 0$. (λ_i can't be negative because $\boldsymbol{\Sigma}_n$ is positive semidefinite.) The λ_i are the eigenvalues and the v_i are the eigenvectors of $\boldsymbol{\Sigma}_n$. Since $\boldsymbol{\Sigma}_n$ is symmetric, the eigenvectors v_i are orthogonal.

From linear algebra, we know that

$$\max_{\boldsymbol{V}\,:\,\boldsymbol{V}^*\boldsymbol{V}=\boldsymbol{I}_k} \operatorname{trace}(\boldsymbol{V}^*\boldsymbol{\Sigma}_n\boldsymbol{V}) = \sum_{i=1}^{k} \lambda_i,$$

and *moreover*, the maximizer $\boldsymbol{V}$ is the one given by $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]$. Hence, these particular $\boldsymbol{v}_j$ give us the desired optimal orthonormal basis for our data $\boldsymbol{x}_i$.

2.4 Intuition for PCA

PCA takes the eigenvector decomposition of $\boldsymbol{\Sigma}_n$ and analyzes the projection of the centered data points ("centered" = subtract sample mean $\boldsymbol{\mu}_n$) on the k top eigenvectors of the sample covariance matrix $\boldsymbol{\Sigma}_n$ as *principal components*. ("k top eigenvectors" = the eigenvectors associated with the largest k eigenvalues.)

2.5 Cost for PCA

The cost of this PCA procedure without using SVD is as follows. We need $\mathcal{O}(nd^2)$ operations to construct $\boldsymbol{\Sigma}_n$, and if you do the eigenvector decomposition in a traditional, naive sense, you need $\mathcal{O}(d^3)$ operations to find $\boldsymbol{V}$. However, the cost is a little bit cheaper via SVD, which we explain below.

Let $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ and

$$\mathbf{1_n} := \left.\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}\right\} \text{(n times)}.$$

Then $\boldsymbol{\Sigma}_n = \frac{1}{n}\left(\boldsymbol{X} - \boldsymbol{\mu}_n \mathbf{1}_{\mathbf{n}}^*\right)\left(\boldsymbol{X} - \boldsymbol{\mu}_n \mathbf{1}_{\mathbf{n}}^*\right)^*$. Now, the idea for saving computational cost is to just compute the SVD of $\boldsymbol{X} - \boldsymbol{\mu}_n \mathbf{1}_{\mathbf{n}}^*$.

The left singular vectors of $\boldsymbol{A} := \boldsymbol{X} - \boldsymbol{\mu}_n \mathbf{1}_{\mathbf{n}}^*$ are the same as the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^* = \boldsymbol{\Sigma}_n$, i.e., they are $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. Therefore, the new cost via SVD is $\mathcal{O}(\min\{n^2 d, nd^2\})$.

Now, from the *full* SVD we get *all* of $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. But we really only want $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$, with $k < n$ (or even $k \ll n$). Computing only the top k singular vectors can be done in $\mathcal{O}(dnk)$ operations. In MATLAB, we can do this with the command `svds`[1]. This is much faster than computing the full SVD. This computation of only the top k singular vectors is done via *Lanczos-type methods*[2].

We also note that *randomized SVD algorithms* can reduce this cost further to $\mathcal{O}\left(nd\log(k) + (n+d)k^2\right)$. We will discuss this later.

2.6 Another optimality property of the SVD

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ and let $\boldsymbol{A} = \sum_{i=1}^n \sigma_i \boldsymbol{u}_i \boldsymbol{w}_i^*$. Denote $\boldsymbol{A}_k = \sum_{i=1}^k \sigma_i \boldsymbol{u}_i \boldsymbol{w}_i^*$ for $k < n$.

Given $\boldsymbol{A}$, then for any matrix $\boldsymbol{B}$ of rank at most k, we have the following *best approximation* result:

$$\|\boldsymbol{A} - \boldsymbol{A}_k\|_{\mathrm{op}} \leq \|\boldsymbol{A} - \boldsymbol{B}\|_{\mathrm{op}},$$

and $\|\boldsymbol{A} - \boldsymbol{A}_k\|_{\mathrm{op}} = \sigma_{k+1}$. In other words, $\boldsymbol{A}_k$ is the *best rank-k approximation to $\boldsymbol{A}$*.

References

[1] G.H. Golub and C.F. van Loan. *Matrix Computations.* Johns Hopkins, Baltimore, third edition, 1996.

[1]Note: inside the code `svds`, MATLAB actually runs the eigenvector decomposition code `eigs` in a smart way.

[2]This has something to do with using power methods to compute the top eigenvector very quickly. Note: Google's PareRank algorithm computes the top eigenvector from a huge matrix when ranking websites with respect to a key word search.