Viet Nam National University Ho Chi Minh City
University of Information Technology

A REPORT ON

SINGULAR VALUE DECOMPOSITION AND PRINCIPAL COMPONENT ANALYSIS

CS115.L11.KHTN

SUBMITTED BY

19521300 Cuong Manh Do Nguyen
19520218 Phu Minh Nguyen
19522424 Trung Huu Le

UNDER THE GUIDANCE OF

Ph.D. HOANG NGOC LUONG

Ho Chi Minh City, January 2021

Contents

Chapter 1

Singular Value Decomposition (SVD)

In Linear Algebra, we learned about diagonalization: a square matrix $A \in R^{n \times n}$ is diagonalizable if exist an invertible matrix P and a diagonal matrix D so that:

$$\mathbf{A} = \mathbf{PDP}^{-1}$$

But it only happens in the square matrix, not all kinds of the matrix. So **Singular Value Decomposition** (SVD) is a ***special matrix factorization*** method for *any* real matrix, which helps us a lot in working with matrix and big data: storing matrix, finding features in data reduction, dimensional reduction, etc. It's considered as a foundation of Machine Learning, one of the most useful tools in numerical linear algebra numerical for data processing. It's also the basis of Principal Component Analysis (PCA) - a wide technique for analyzing high dimensional data. SVD is the basis of the facial recognition algorithm.

1.1 Introduction to SVD

The singular value decomposition of a matrix is usually referred to as the SVD. This is the final and best factorization of any matrix:

$$\mathbf{A}_{m \times n} = \mathbf{U}_{m \times m} \mathbf{\Sigma}_{m \times n} (\mathbf{V}_{n \times n})^T \qquad (1)$$

where U is **orthonormal**, Σ is almost a **diagonal** matrix - only contains real numbers ordered **descending** $\sigma_{1,2,....m}$ in main **diagonal** line ,V is also **orthonormal**.

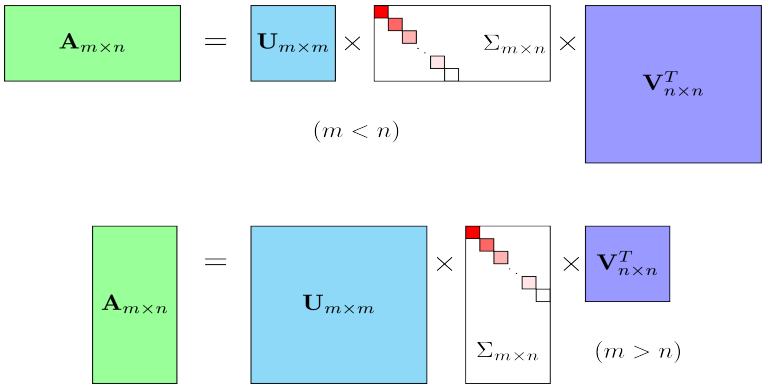


Figure 1.1: The image describes the SVD for matrix A in 2 cases: $m < n$ and $m > n$

1.2 Math Revision

1.2.1 Orthogonal matrix and orthonormal vector

- We say that 2 vectors are orthogonal if they are non-zero vectors and perpendicular to each other. i.e their dot product equal to zero. For example, u and v are orthogonal if $u^T v = 0$.
- We say that a set of vectors $\{\vec{u_1}, \vec{u_2}, ..., \vec{u_n}\}$ is mutually orthogonal if every pair of vectors is orthogonal.
- 2 vectors are orthonormal if they are orthogonal and their magnitude equal to 1.
- A set of vectors $\{\vec{u_1}, \vec{u_2}, ..., \vec{u_n}\}$ is orthonormal if every pair of vectors is orthonormal. i.e.

$$u_i^T u_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

1.2.2 Gram - Schmidt Algorithm

For a set of linearly independent vectors $\{\vec{v_1}, \vec{v_2}, ..., \vec{v_n}\} \in \mathbb{R}^n$, we can construct an orthonormal set of vectors:

- Let $u_1 = \frac{v_1}{\|v_1\|}$
- For $i = 2$ to n: * Orthogonalization: $u_i = v_i - (u_{i-1}^T v_i)u_{i-1} - ... - (u_1^T v_i)u_1$
- Normalization: $u_i = \frac{u_i}{\|u_i\|}$

For example: Matrix $A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ with a set of vectors $a_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $a_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

- Check independently matrix: $det(A) = -2 \neq 0$

- Let $u_1 = \frac{a_1}{\|a_1\|} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$
- $i = 2$: $u_2 = a_2 - (u_1^T a_2)u_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$
- $u_2 = \frac{u_2}{\|u_2\|} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$

So matrix A after orthonormalization: $A = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$

1.3 Calculating SVD

Ignoring dimension of the matrix, base on equation (1) we can compute AA^T like this:

$$\mathbf{A}\mathbf{A}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T\mathbf{U}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T\mathbf{U}^{-1} \qquad (2)$$

$\Sigma\Sigma^T$ is a diagonal matrix contains $\sigma_1^2, \sigma_2^2 ... \sigma_m^2$. These σ^2 is eigenvalues of AA^T and its is not negative. Those σ_j is square root of AA^T eigenvalues - also called as **singular values**. Column vectors of U are eigenvectors of AA^T. We call these vectors are **left-singular vectors**.

In the other hand, calculating A^TA will look like this:

$$\mathbf{A}^T\mathbf{A} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^{-1} \qquad (3)$$

Similar to AA^T, column vectors of V are eigenvectors of A^TA. We call these vectors are **right-singular vectors**.

Prove eigenvalues of A^TA and AA^T are the same:

Let $\lambda \neq 0$ be an eigenvalue of A^TA with corresponding eigenvector $\mathbf{v} \neq \mathbf{0}$:

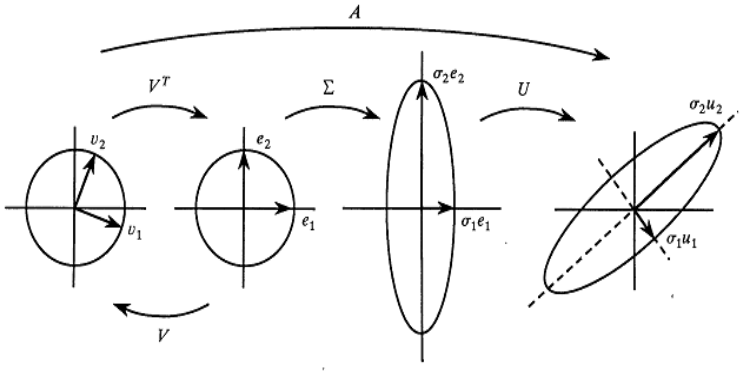
$$\begin{aligned} \Rightarrow A^TA\mathbf{v} &= \lambda\mathbf{v} \\ \Leftrightarrow AA^TA\mathbf{v} &= \lambda A\mathbf{v} \\ \Leftrightarrow AA^T\mathbf{u} &= \lambda u \quad (u = Av) \end{aligned}$$

λ is an eigenvalue of AA^T as well, with eigenvector $\mathbf{u} = A\mathbf{v}$.

Now show that if λ s eigenvalue for AA^T with corresponding eigenvector $\mathbf{u} \neq \mathbf{0}$

$$\begin{aligned} \Rightarrow AA^T\mathbf{u} &= \lambda\mathbf{u} \\ \Leftrightarrow A^TAA^T\mathbf{u} &= \lambda A^T\mathbf{u} \\ \Leftrightarrow A^TA\mathbf{z} &= \lambda\mathbf{z} \quad (z = A^Tu) \end{aligned}$$

So eigenvalues of A^TA and AA^T are the same! (4)
From (2)(3)(4), we can get matrix Σ, U, V just by diagonalizing A^TA or AA^T.
Image depicting the transfer facility in svd:



1.4 Some special SVD

In reality, data is very complex, and it can be lots of entries and features. For example, human genetic data can be a sequence of thousand features, and there are thousands of people - which is huge; finding "eigenface" of people for facial recognition, a grey picture contains 1 face, may have a size 400 x 400 pixel, having different lighting condition, enormous data like that can't be interpreted and analyzed. Even though, after doing SVD with these data, it's still large. Truncated SVD will help us solve this big data problem. We will store a part of the origin SVD, but still knowing the main features.

1.4.1 Compacted SVD

Equation (1) can be written as below:

$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

with every $u_i v_i^T, 1 \leq i \leq r$ is a rank-1 matrix.

In this formula, A only depends on **first r columns** of U, **first r rows** of V and **r** values in main diagonal line of Σ. So we have a better decomposition called **compacted SVD**:

$$\mathbf{A} = \mathbf{U}_r \Sigma_r (\mathbf{V}_r)^T$$

If our A has a smaller rank than **columns and rows** of A, meaning $r \ll m, n$. Therefore, we have the benefit of compacted SVD in storing data.

1.4.2 Truncated SVD

Remembered, σ values in the main diagonal line of Σ is non-zero and ordered descending. In common, some first values of σ_i are large, remaining values are small and can be zero. We do not want to store all the values in the SVD. Then, we can approximate matrix $A \approx \hat{A}$ equal to sum of k ($< r$) rank-1 matrices:

$$\mathbf{A} \approx \hat{\mathbf{A}} = U_k \Sigma_k V_k^T = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T$$

Pushing away $r-k$ small and non-zero values in **SVD** is called **Truncated SVD**. The error of subtraction $A - A_k$ is calculated by the Frobineous norm of the subtraction. But we have a theorem for it. **The error will equal to total square of the cut-off eigenvalues in truncated SVD**.

$$||\mathbf{A} - \mathbf{A}_k||_F^2 = \sum_{i=k+1}^{r} \sigma_i^2 \quad (5)$$

Proof:

$$\begin{aligned}
||\mathbf{A} - \mathbf{A}_k||_F^2 &= || \sum_{i=k+1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T ||_F^2 \qquad (6) \\
&= \text{trace} \left\{ \left(\sum_{i=k+1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \right) \left(\sum_{j=k+1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \right)^T \right\} \\
&= \text{trace} \left\{ \sum_{i=k+1}^{r} \sum_{j=k+1}^{r} \sigma_i \sigma_j \mathbf{u}_i \mathbf{v}_i^T \mathbf{v}_j \mathbf{u}_j^T \right\} \\
&= \text{trace} \left\{ \sum_{i=k+1}^{r} \sigma_i^2 \mathbf{u}_i \mathbf{u}_i^T \right\} \\
&= \text{trace} \left\{ \sum_{i=k+1}^{r} \sigma_i^2 \mathbf{u}_i^T \mathbf{u}_i \right\} \\
&= \text{trace} \left\{ \sum_{i=k+1}^{r} \sigma_i^2 \right\} \\
&= \sum_{i=k+1}^{r} \sigma_i^2
\end{aligned}$$

With $k = 0$, we got:

$$||\mathbf{A}||_F^2 = \sum_{i=1}^{r} \sigma_i^2 \quad (7)$$

From (5)(7) we can infer:

$$\frac{||\mathbf{A} - \mathbf{A}_k||_F^2}{||\mathbf{A}||_F^2} = \frac{\sum_{i=k+1}^{r} \sigma_i^2}{\sum_{j=1}^{r} \sigma_j^2} \quad (8)$$

Thus, **the error of this approximation is very small if the cut-off eigenvalues is negligible for comparing to first k eigenvalues**. The theorem (4) is important for calculating how much information we want to store. From equation (7), we can pick smallest k for storing up to $y\%$ information. We can call it a low-rank approximation.

Best k-rank approximation

This course: SVD - Princeton Course proves that $B = A_k$ is also an optimal value in this optimization problem:

$$\min_{\mathbf{B}} ||\mathbf{A} - \mathbf{B}||_F \quad \text{s.t.} \quad \text{rank}(\mathbf{B}) = k$$

In above proof, $||\mathbf{A}-\mathbf{A}_k||_F^2 = \sum_{i=k+1}^{r} \sigma_i^2$ (4). If we using 2-norm instead of Frobeneous norm (F-norm) to calculate the error, A_k is also an optimal value for that optimization problem:

$$\min_{\mathbf{B}} ||\mathbf{A} - \mathbf{B}||_2 \quad \text{s.t.} \quad \text{rank}(\mathbf{B}) = k$$

1.5 Step by step computation Of SVD

But in practice, we do not compute both equation (2)(3). We choose 1 equation, compute U or V and Σ, then using the definition of SVD (1) to compute the leftover. Example:

$$A = \begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix}$$

We're using equation (3)

$$A^T A = \begin{pmatrix} 5 & -1 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix} = \begin{pmatrix} 26 & 18 \\ 18 & 74 \end{pmatrix}$$

Diagonalizing A^TA, we got Σ, V.

$$\begin{aligned} det(A^TA - \lambda I) &= 0 \\ \lambda^2 - 100\lambda + 1600 &= 0 \\ (\lambda - 20)(\lambda - 80) &= 0 \\ \lambda &= 80 \quad or \quad \lambda = 20 \end{aligned}$$

- $With\, \lambda_1 = 80$

$$\begin{aligned} A^TA - 80I &= \begin{pmatrix} -54 & 18 \\ 18 & -6 \end{pmatrix} \\ v_1 &= \begin{bmatrix} 1 \\ 3 \end{bmatrix} \end{aligned}$$

- $With\, \lambda_2 = 20$

$$\begin{aligned} A^TA - 20I &= \begin{pmatrix} 6 & 18 \\ 18 & 54 \end{pmatrix} \\ v_2 &= \begin{bmatrix} -3 \\ 1 \end{bmatrix} \end{aligned}$$

But $v1, v2$ is orthogonal already. Normalizing v_1, v_2 .So now, we got:

$$\Sigma = \begin{pmatrix} 4\sqrt{5} & 0 \\ 0 & 2\sqrt{5} \end{pmatrix}, V = \begin{pmatrix} 1/\sqrt{10} & -3/\sqrt{10} \\ 3/\sqrt{10} & 1/\sqrt{10} \end{pmatrix}$$

Now we use the SVD definition:

$$A = U\Sigma V^T$$
$$AV = U\Sigma$$

$$AV = \begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix} \begin{pmatrix} 1/\sqrt{10} & -3/\sqrt{10} \\ 3/\sqrt{10} & 1/\sqrt{10} \end{pmatrix} = \begin{pmatrix} 2\sqrt{10} & -\sqrt{10} \\ 2\sqrt{10} & \sqrt{10} \end{pmatrix} = U\Sigma$$

$$=> U = AV\Sigma^{-1} = \begin{pmatrix} 2\sqrt{10} & -\sqrt{10} \\ 2\sqrt{10} & \sqrt{10} \end{pmatrix} \begin{pmatrix} \sqrt{5}/20 & 0 \\ 0 & \sqrt{5}/10 \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

Conclusion, we got 3 matrix U, Σ, V as a result after SVD-factorization matrix A

$$A = \begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix} = U\Sigma V^T$$

$$U = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}, \Sigma = \begin{pmatrix} 1/\sqrt{10} & -3/\sqrt{10} \\ 3/\sqrt{10} & 1/\sqrt{10} \end{pmatrix}, V = \begin{pmatrix} 1/\sqrt{10} & -3/\sqrt{10} \\ 3/\sqrt{10} & 1/\sqrt{10} \end{pmatrix}$$

1.6 Application of SVD

Image Compression

We will jump into a real example of image compression using SVD. We have a high-resolution image, but storing the full size of it will consume memory. So we compress the picture, but we can still recognize the picture. The picture still has its main features.

Figure 1.2: Demo picture

We will convert the image to grayscale.

Figure 1.3: Grayscale demo picture converted

```
from numpy import linalg as LA
U, S, Vt = LA.svd(gray)
```

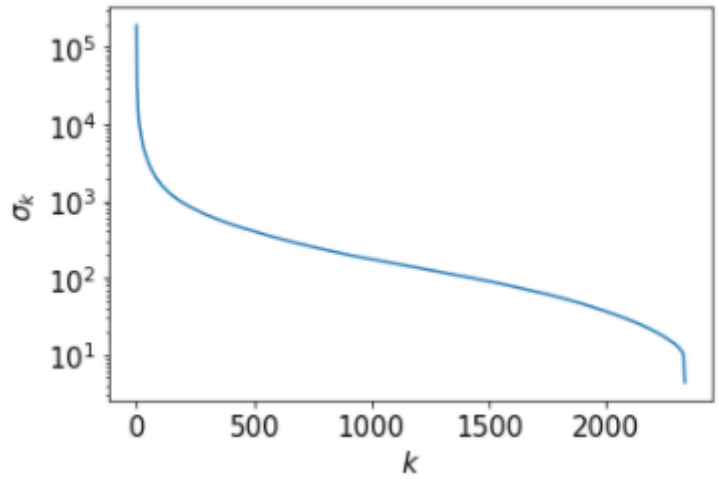


Figure 1.4: The variability of singular values

We can see that the singular values decrease quickly at $k = 100$.

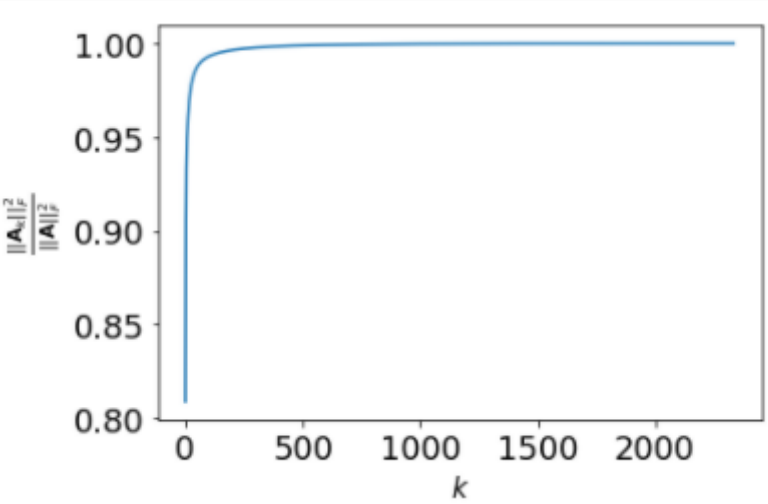


Figure 1.5: Ratio keep image features

Above graph represent for ratio keep information from the picture when k change. We can see that at $k = 100$, we can keep almost all the information from the picture.

Figure 1.6: Picture got from different k

Storing picture with Truncated SVD, we will store matrices $\mathbf{U}_k \in \mathbb{R}^{m\times k}, \Sigma_k \in \mathbb{R}^{k\times k}, \mathbf{V}_k \in \mathbb{R}^{n\times k}$. Total elements we need to store is $k(m+n+1)$ - notice that we only store K values in the main diagonal line in Σ_k. Assuming an float element cost 4 bytes, total bytes is $4k(m+n+1)$, original picture cost mn bytes - intergers cost 1 byte. So compression rate is :

$$\frac{4k(m+n+1)}{mn}$$

When $k \ll mn$, we got a rate smaller than 1. With our example,choosing $k = 100$,$m = 3500, n = 2336$. We got a rate approximately 0.29, saving 70% memory.

Chapter 2

Principal Component Analysis (PCA)

Like Truncated SVD, PCA is a method that reduces the dimension of data and keeps its main features. PCA convert the basis of related variables to the basis of unrelated variables and maximize the variance.

2.1 Some probability theory and statistics

2.1.1 Expected value and Variance

In probability theory:

Expected value of a random variable is the mean of all specific values of it.

Variance is the expectation of the squared deviation of a random variable from its mean.

In 1-dimension data:

Given N values $x_1, x_2, \ldots, x_N$. Expected value and variance of the data calculated by follow formula:

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

$$var(X) = \sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})^2$$

Expected value is simply arithmetic mean of all value in the data. σ is standard deviation.

In N-dimension data:

Given N point of data expressed by column vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$, then our **expected**

value vector is caculated from this formula:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$$

2.1.2 Covariance and Covariance matrix

In probability theory:

Covariance is a measure of the relationship between random variables.

$$cov(x, y) = \frac{\sum_{i=1}^{n}(x_i - \hat{x})(y_i - \hat{y})}{N}$$

Covariance matrix is a square matrix with the variance of variables in the diagonal line and the covariance of variables in the other position.

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T = \frac{1}{N}\hat{\mathbf{X}}\hat{\mathbf{X}}^T \ (with\ \hat{\mathbf{X}} = \mathbf{x}_n - \bar{\mathbf{x}})$$

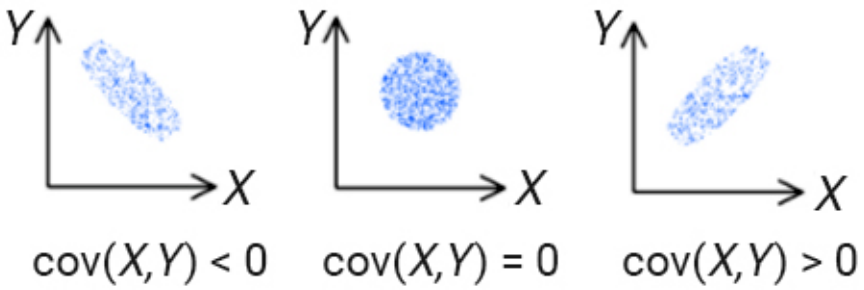
Example for covariance matrix of 3 features X, Y, Z:

$$S = \begin{pmatrix} var(X) & cov(X,Y) & cov(X,Z) \\ cov(X,Y) & var(Y) & cov(Y,Z) \\ cov(X,Y) & cov(Y,Z) & var(Z) \end{pmatrix}$$

Some features about covariance matrix:

- Every value in the main diagonal line in the covariance matrix is a non-negative value. It's also a variance in each dimension of the data.
- Remaining values that are not in the main diagonalize express correlation between the data components also called covariance. These value can be positive, negative or 0.
- We got data that does not correlate with its components or dimension if we got a diagonal covariance matrix.

An example image about correlation or uncorrelation data:



The reason to choose covariance matrix to get the principal components of features X in PCA: The covariance matrix represents the variation in the data set. The diagonal elements show the scatter of the variables and the correlation between the variables in the other elements. For variables whose value does not change or change insignificantly, we consider that variable does not store too much information. Thus, PCA is converting the original base system to a new one of uncorrelated variables and eliminating the variables with small variances to optimize the amount of information stored. PCA helps reduce the number of initial variables, into some important components, while preserving data.

2.2 Into PCA

The idea of PCA is plotting the data into a new basis system. In that system, the importance of the components is significantly different from ignoring the least important component.

From the data X, we will split X into A and B in which the data is mainly concentrated in A, B only carries a small amount of information.

Assume the new orthonormal basis is U, and we want to keep K points on this new basis. First, K points plot almost information in the data X. This idea is shown in the below image:

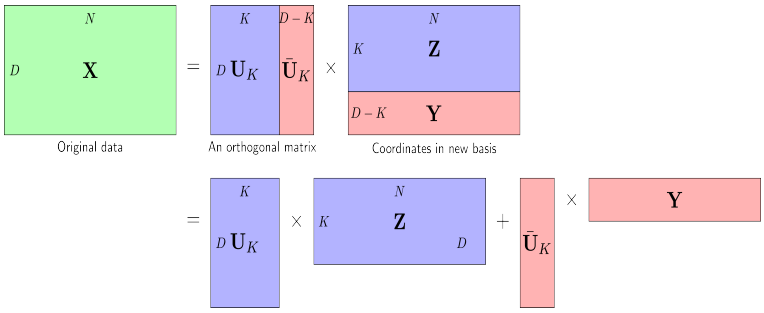


Figure 2.1: The PCA idea

We having:

$$\begin{aligned} \mathbf{X} &= \mathbf{U}_K\mathbf{Z} + \bar{\mathbf{U}}_K\mathbf{Y} \\ &= \mathbf{A} + \mathbf{B} \end{aligned}$$

with $\mathbf{A} = \mathbf{U}_K\mathbf{Z}, \mathbf{B} = \bar{\mathbf{U}}_K\mathbf{Y}$. Assuming we have A, minimizing the loss, we approximate B with a formula :

$$B = \bar{\mathbf{U}}_K\bar{\mathbf{U}}_K^T\bar{\mathbf{x}}\mathbf{1}^T$$

We can watch a proof in a blog about basic ML by Vu Huu Tiep PCA - machine-learningcoban. We got:

$$\mathbf{X} \approx \tilde{\mathbf{X}} = \mathbf{U}_K\mathbf{Z} + \bar{\mathbf{U}}_K\bar{\mathbf{U}}_K^T\bar{\mathbf{x}}\mathbf{1}^T$$

From above formula, we have our loss function:

$$\begin{aligned}
L &= \frac{1}{N}||\mathbf{X} - \tilde{\mathbf{X}}||_F^2 = \frac{1}{N}||\bar{\mathbf{U}}_K\bar{\mathbf{U}}_K^T\mathbf{X} - \bar{\mathbf{U}}_K\bar{\mathbf{U}}_K^T\bar{\mathbf{x}}\mathbf{1}^T||_F^2 \\
&= \frac{1}{N}||\bar{\mathbf{U}}_K\bar{\mathbf{U}}_K^T\mathbf{X} - \bar{\mathbf{x}}\mathbf{1}^T||_F^2 \\
&= \frac{1}{N}||\bar{\mathbf{U}}_K^T(\mathbf{X} - \bar{\mathbf{x}}\mathbf{1})^T||_F^2 \quad (\bar{\mathbf{U}}_K \ orthonormal) \\
&= \frac{1}{N}||\hat{\mathbf{X}}^T\bar{\mathbf{U}}_K||_F^2 = \frac{1}{N}||\bar{\mathbf{U}}_K^T\hat{\mathbf{X}}||_F^2 \\
&= \frac{1}{N}\sum_{i=K+1}^{D}||\hat{\mathbf{X}}^T\mathbf{u}_i||_2^2 \\
&= \sum_{i=K+1}^{D}\mathbf{u}_i^T\mathbf{S}\mathbf{u}_i \quad (S = \hat{\mathbf{X}}\hat{\mathbf{X}}^T)
\end{aligned}$$

After that, we optimize the loss function L :

$$\begin{aligned}
L &= \sum_{i=k+1}^{D}\mathbf{u}_i^T\mathbf{S}\mathbf{u}_i = \frac{1}{N-k}||\hat{\mathbf{X}}^T\bar{\mathbf{U}}_K||_F^2 \\
&= \frac{1}{N-k}\text{trace}(\hat{\mathbf{X}}^T\bar{\mathbf{U}}_K\bar{\mathbf{U}}_K^T\hat{\mathbf{X}}) \quad (\bar{\mathbf{U}}_K \ orthonormal) \\
&= \frac{1}{N-k}\text{trace}(\hat{\mathbf{X}}^T\hat{\mathbf{X}}) = \frac{1}{N-k}\text{trace}(\hat{\mathbf{X}}\hat{\mathbf{X}}^T) \\
&= \text{trace}(\mathbf{S}) = \sum_{i=k+1}^{D}\lambda_i \ (With \ \lambda_i \geq 0)
\end{aligned}$$

L do not depend on U. Function L minimized $\Leftrightarrow$From $k+1$ to D, we have $D-k$ minimum eigenvalues of the covariance matrix. In the other words, we have the k biggest eigenvalues of the covariance matrix S and u_i is the k eigenvectors correspondingly to those eigenvalues.
Conclusion, PCA helps us to reduce dimension the basis system for the original data but still retain most of the information. (From D to K with $K \ll D$).

2.3 Step By Step Computation Of PCA

Having 7 steps to compute of PCA

Step 1: Find mean vector

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

Step 2: Subtract mean

$$\hat{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$$

Step 3: Calculate the covariance matrix for the features in the dataset

$$\mathbf{S} = \frac{1}{N}\hat{\mathbf{X}}\hat{\mathbf{X}}^T$$

Step 4: Calculate eigenvalues, eigenvector and sort those eigenvalues in ordered descending (These eigenvector must be normalized)

Step 5: Pick K eigenvector u corresponding to the K highest eigenvalues

Step 6: Project data to selected eigenvector

$$\mathbf{Z} = \mathbf{U}_K^T\hat{\mathbf{X}}$$

We can approximate the original data by doing:

$$\mathbf{x} \approx \mathbf{U}_K\mathbf{Z} + \bar{\mathbf{x}}$$

For large datasets, we need to add the data normalization step before performing the above steps. Skipping the data normalization step can greatly affect the results.
It can be calculated like so:

$$x = \frac{variable\ value - mean}{standard\ deviation}$$

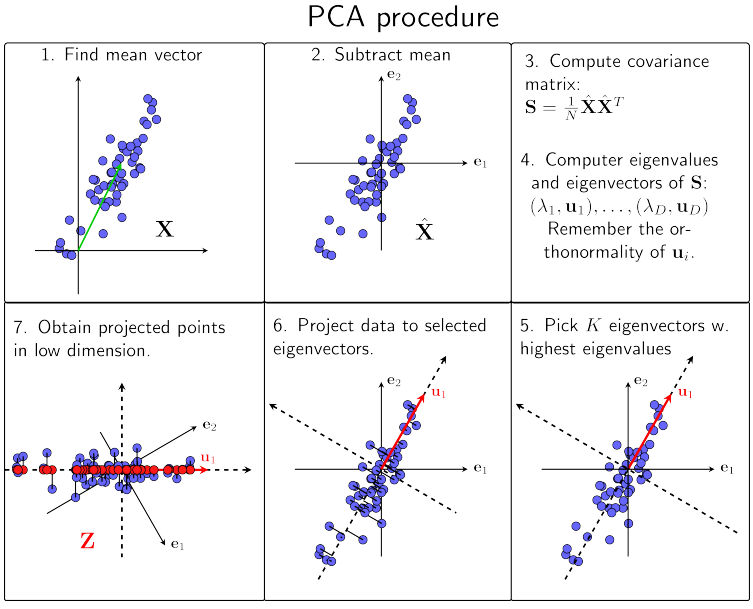


Figure 2.2: Step by step explanation of PCA

Running example PCA:
Given the following data, use PCA to reduce the dimension from 2 to 1

Feature	Example1	Example2	Example3	Example4
x	4	8	13	7
y	11	4	5	14

$$\Rightarrow X = \begin{pmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 5 & 14 \end{pmatrix}$$

Step 1: Find mean vector

- Number of features, m=2
- Number of samples, n=4

$$\begin{aligned} \bar{x} &= \frac{1}{N}\sum_{n=1}^{N} x_n \\ \Leftrightarrow \bar{x} &= \begin{bmatrix} 8 \\ 8.5 \end{bmatrix} \end{aligned}$$

Step 2: Subtract mean

$$\hat{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$$

$$\Rightarrow \hat{X} = \begin{pmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{pmatrix}$$

Step 3: Calculate the covariance matrix for the features in the dataset

$$\begin{aligned} \mathbf{S} &= \frac{1}{N}\hat{\mathbf{X}}\hat{\mathbf{X}}^T \\ &= \begin{pmatrix} var(x) & cov(x,y) \\ cov(y,x) & var(y) \end{pmatrix} \\ &= \begin{pmatrix} 14 & -11 \\ -11 & 23 \end{pmatrix} \end{aligned}$$

Step 4: Calculate eigenvalues, eigenvector and sort those eigenvalues in ordered descending (These eigenvector must be normalized)

$$\begin{aligned} det(S - \lambda I) &= 0 \\ \lambda^2 - 37\lambda + 201 &= 0 \\ \Rightarrow \lambda &= 30.3849 \quad or \quad \lambda = 6.6151 \end{aligned}$$

With $\lambda = 30.3849$ (this is the first principal component)

$$\begin{aligned} (S - \lambda I)u_1 &= 0 \\ \begin{pmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{pmatrix} \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \Rightarrow u_1 &= \begin{bmatrix} 11 \\ -16.3849 \end{bmatrix} \end{aligned}$$

Normalizing u_1 we got:

$$u_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$$

Step 5: Pick K eigenvector w. highest eigenvalues

Because the data has only 2 dimension so we skip this step.

Step 6: Project data to selected eigenvector.

$$p_{11} = u_1^T \begin{bmatrix} -4 \\ 2.5 \end{bmatrix} = -4.3052$$

$$p_{12} = u_1^T \begin{bmatrix} 0 \\ -4.5 \end{bmatrix} = 3.7361$$

$$p_{13} = u_1^T \begin{bmatrix} 5 \\ -3.5 \end{bmatrix} = 5.6928$$

$$p_{14} = u_1^T \begin{bmatrix} -1 \\ 5.5 \end{bmatrix} = -5.1238$$

Step 7: Reducing the dimensions of the data set

$$\mathbf{Z} = \mathbf{U}_K^T \hat{\mathbf{X}}$$

Feature	Example1	Example2	Example3	Example4
PC1	-4.3052	3.7361	5.6928	-5.1238

2.4 Relationship between PCA and SVD

The data has been preprocessed to have zero mean.

$$\Rightarrow S = \frac{1}{n}\mathbf{X}\mathbf{X}^\top$$

Cause S is a symmetric matrix, eigenvectors of S are orthogonal. Therefore, we normalize that eigenvectors and they will be orthonormal.

$$S = \frac{1}{n}\mathbf{U}\mathbf{D}\mathbf{U}^\top \quad (1)$$

On the other hand, when implementing SVD, we have:

$$\begin{aligned} S &= \frac{1}{n}\mathbf{X}\mathbf{X}^\top = \frac{1}{n}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)^\top = \frac{1}{n}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)(\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top) \\ &= \frac{1}{n}\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top \quad (\bar{\mathbf{U}} \textit{ orthonormal}) \quad (2) \end{aligned}$$

From (1) and (2), U after diagonalize covariance matrix is the same as when we factorize matrix by using SVD.

We can approximate matrix by using Truncated SVD:

$$\mathbf{X} \approx \tilde{\mathbf{X}} = \mathbf{U}_K\mathbf{\Sigma}_K\mathbf{V}_K^T \quad (3)$$

When we approximate by using PCA:

$$\mathbf{X} \approx \tilde{\mathbf{X}} = \mathbf{U}_K\mathbf{Z} \quad (4)$$

From (3) and (4), U_k in truncated SVD is the same as U_k in PCA and $\mathbf{\Sigma}_K\mathbf{V}_K^T$ in truncated SVD is Z in PCA.

2.5 PCA for large-scale problems

In practice, the number of data points N, and its dimensions D are so large. Therefore, we have to find eigenvalues for a huge matrix. E.g. we have 1 Million images containing 1000×1000 pixels, so we have $D = 10^6 = N$, and it is difficult and time-consuming to find eigenvalues for a covariance matrix that is calculated from $10^6 \times 10^6$ matrix. However, we have a method to find that approximately eigenvalues quickly called **Power Method**.

To find eigenvalues and eigenvectors for a positive semi-definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$:

- Step 1: Choose a random vector $\mathbf{q}^{(0)} \in \mathbb{R}^n, ||\mathbf{q}^{(0)}||_2 = 1$ and $k = 1$
- Step 2: Calculate $\mathbf{z} = \mathbf{A}\mathbf{q}^{(k-1)}$
- Step 3: Normalize $\mathbf{q}^{(k)} = \frac{\mathbf{z}}{||\mathbf{z}||_2}$
- Step 4: While $||\mathbf{q}^{(k)} - \mathbf{q}^{(k-1)}||_2 < \epsilon$ then stop, else $k = k+1$ and return to step 2.

Now we have largest eigenvalue $\lambda_1 = (\mathbf{q}^{(k)})^T\mathbf{A}\mathbf{q}^{(k)}$ and corresponding eigenvector $\mathbf{q}^{(k)}$. Let prove this method: If A is a positive semi-definite matrix then there exist n independent eigenvectors of A. Let $x_1, ..., x_n$ be these eigenvectors, then $x_1, ..., x_n$ form a basis of $\mathbb{R}^n$. Hence the initial vector $q^{(0)}$ can be written as: $q^{(0)} = a_1x_1 + a_2x_2 + ... + a_nx_n$ where $a_1, ..., a_n$ are scalars.

Multiplying both sides of the equation in A^k yields:

$$\begin{aligned} A^k q^{(0)} &= A^k(a_1x_1 + a_2x_2 + ... + a_nx_n) \\ &= a_1A^kx_1 + a_2A^kx_2 + ... + a_nA^kx_n \\ &= a_1\lambda_1^kx_1 + a_2\lambda_2^kx_2 + ... + a_n\lambda_n^kx_n \\ &= a_1\lambda_1^k(x_1 + \sum_{j=2}^{n} \frac{a_j}{a_1}(\frac{\lambda_j}{\lambda_1})^kx_j) \end{aligned}$$

If $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$ then we say that λ_1 is a dominant eigenvalue. In this case $(\frac{\lambda_j}{\lambda_1})^k \to 0$ and therefore if $a_1 \neq 0, A^kq^{(0)} \to a_1\lambda_1^kx_1$. The power method normalizes the products $Aq^{(k-1)}$ to avoid overflow, therefore it converges to x_1.

To find the next eigenvalue and eigenvector, we have a matrix $\mathbf{B} = \mathbf{A} - \lambda_1\mathbf{v}_1\mathbf{v}_1^T$ that have eigenvalues $\lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n \geq 0$ and corresponding eigenvectors $\mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_n$.

Prove it by using inductive method:

- With $i = 1$:

$$\mathbf{B}\mathbf{v}_1 = (\mathbf{A} - \lambda_1\mathbf{v}_1\mathbf{v}_1^T)\mathbf{v_1} = \mathbf{A}\mathbf{v}_1 - \lambda_1\mathbf{v}_1 = \mathbf{0}$$

- With $i > 1$:

$$\mathbf{B}\mathbf{v}_i = (\mathbf{A} - \lambda_1\mathbf{v}_1\mathbf{v}_1^T)\mathbf{v}_i = \mathbf{A}\mathbf{v}_i - \lambda_1\mathbf{v}_1(\mathbf{v}_1^T\mathbf{v}_i) = \mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i$$

Hence, $(\lambda_2, \mathbf{v}_2)$ has became a large pair of eigenvalue and eigenvector of $\mathbf{B}$. Therefore, we can find this pair by using Power method for $\mathbf{B}$. Repeat this process, we can find up to eigenvalue $\mathbf{K} - th$ for covariance matrix.

2.6 Some notice when calculating PCA in real data

There are two cases that we need to notice when calculating PCA in reality. First, the data is much smaller than the dimension of the data. Second, the training data is huge, can up to millions. The covariance matrix and eigenvalue calculation can be impossible. We still have solutions for those cases.

This section assumes that the data has been normalized, which the expected vector subtracts the data. Then, the covariance matrix will be calculated by this formula: $\mathbf{S} = \frac{1}{N}\mathbf{X}\mathbf{X}^T$.

2.6.1 Dimensions of the data is bigger than datapoints

When $D > N$, eigenvalues of the covariance matrix S will not larger than N. So we need to pick $K \leq N$ cause we can't pick $K > N$ non-zero eigenvalue of a rank-N matrix.

Calculating eigenvalues and eigenvectors can be be effectively implemented if we follow these features:

- Eigenvalues of A is equal to eigenvalues of kA with non-zero k
- Eigenvalues of $\mathbf{AB}$ is equal to eigenvalues of $\mathbf{AB}$ with $\mathbf{A} \in \mathbb{R}^{d_1 \times d_2}, \mathbf{B} \in \mathbb{R}^{d_2 \times d_1}$ and d_1, d_2 is a non-zero number.

  Instead of finding eigenvalue in covariance matrix $\mathbf{S} \in \mathbb{R}^{D \times D}$, we can find eigenvalue matrix $\mathbf{T} = \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{N \times N}$ having smaller dimension ($N < D$)
- Assuming $\mathbf{T} = \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{N \times N}$ is a pair of eigen value-vector of $\mathbf{T}$ then $(\lambda, \mathbf{Xu})$is a pair of eigen value-vector of $\mathbf{S}$. Proof:

$$\mathbf{X}^T\mathbf{Xu} = \lambda\mathbf{u} \Rightarrow (\mathbf{XX}^T)(\mathbf{Xu}) = \lambda\mathbf{Xu}$$

2.6.2 Normalization EigenVectors

Recall the definition of eigenspace: The eigenspaces corresponding to the eigenvalues of a matrix are the span subspace created by all the eigenvectors corresponding to that eigenvalue.

The last thing to do in PCA is normalizing eigenvectors so that they form an orthonormal system. This can be based on the following two points:

- First,if $\mathbf{A}$ is a symmetric matrix, $(\lambda_1, \mathbf{x}_1), (\lambda_2, \mathbf{x}_2)$ is pairs of eigenvalue-eigenvector of A, then $\mathbf{x}_1^T\mathbf{x}_2 = 0$. In other words, any two vectors in two different eigenspaces of a symmetrical matrix are perpendicular to each other. Proof:

$$\mathbf{x}_2^T\mathbf{A}\mathbf{x}_1 = \mathbf{x}_1^T\mathbf{A}\mathbf{x}_2 = \lambda_1\mathbf{x}_2^T\mathbf{x}_1 = \lambda_2\mathbf{x}_1^T\mathbf{x}_2 \Rightarrow \mathbf{x}_1^T\mathbf{x}_2 = 0 \qquad cause \quad \lambda_1 \neq \lambda_2$$

- Second, with independent eigenvalues found in eigenspaces, we can use the Gram-Schmit process to normalize them into an orthonormal system.

Combining two points above, we can obtain the individual vectors to form an orthonormal system, which is the matrix $\mathbf{U}_K$ in PCA.

2.7 Application and Example of PCA

2.7.1 PCA in analysis data

PCA is a great tool for analyzing high-dimensional data. We gonna create 2d data with Gaussian distribution. So we can try to analyze the data for understanding the dominant direction of variants in that data.

```
import numpy as np

xC = np.array([2.5,0.5]) #center of data (mean)
sig = np.array([2.5,0.5]) #principal axes

theta = np.pi/3

R = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta),np.cos(theta)]])  #rotaion matrix
nPoints = 10000  #creating 10k points
X = R .dot(np.diag(sig)).dot(np.random.randn(2,nPoints))
+ np.diag(xC).dot(np.ones((2,nPoints)))
```

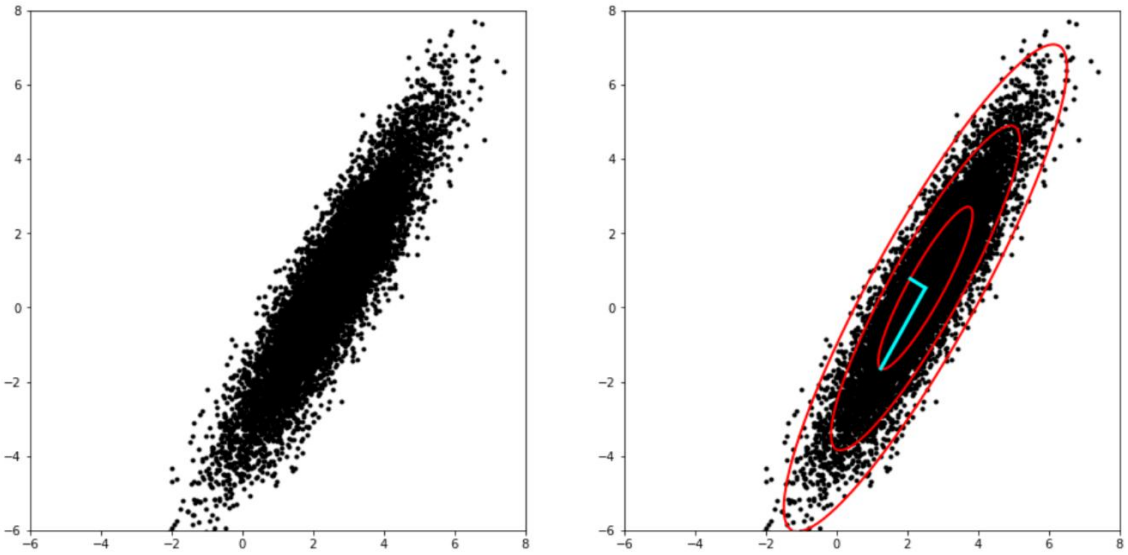


Figure 2.3: Plotting 10000 points and its standard deviation.

After plotting the data and its principal components. We can see singular values in Σ almost 2.5 and 0.5. The first principal components direction has a lot of variance of 2 and the second components have a small variance of 0.5. In the U matrix, we can know how the data is rotated. The first column in U is the rotated first principal components direction, the same with the second column. 3 ellipse is represented for 3 standard deviation ellipse. We can know how much standard deviation a data point has.

2.7.2 Eigenfaces for face recognition

Eigenfaces is one of the most popular methods for facing recognition. This method's idea is finding a vector space that has a small dimension than the original space. Then,

we can use these vectors as the feature vectors to projecting data for classification. Actually, eigenfaces are eigenvectors corresponding to the largest eigenvalues of the covariance matrix in PCA. As an example, we use the Labeled Faces in the Wild dataset to do an eigenfaces test.

The picture below is the first 10 pictures in the dataset.

Figure 2.4: First 10 pictures in the dataset

The dataset that we used has 1288 samples and 1850 dimensions. This means every sample has the size of 50×37.

```
from sklearn.decomposition import PCA
n_components = 150
pca = PCA(n_components, svd_solver='randomized',whiten=True)
pca.fit(X)
eigenfaces = pca.components_.reshape((n_components, h, w))
U = pca.components_.T
```

Figure 2.5: Eigenface mean

Figure 2.6: First 30 eigenfaces

This is the result after we were projecting data on eigenfaces space and approximating that.

Figure 2.7: First 5 face approximated on eigenfaces

2.7.3 Eigengene in human gene data

We gonna analyze an Ovarian cancer data (a built data) - 216 individual patients with 4000 genetic markers that are measured for every patient. These patients are broken into two groups. The first half has cancer, and the second half has no cancer. We are gonna use PCA to decompose this high dimensional data. After that, we are gonna visualize the data in 3D, find the relationship of the variables.

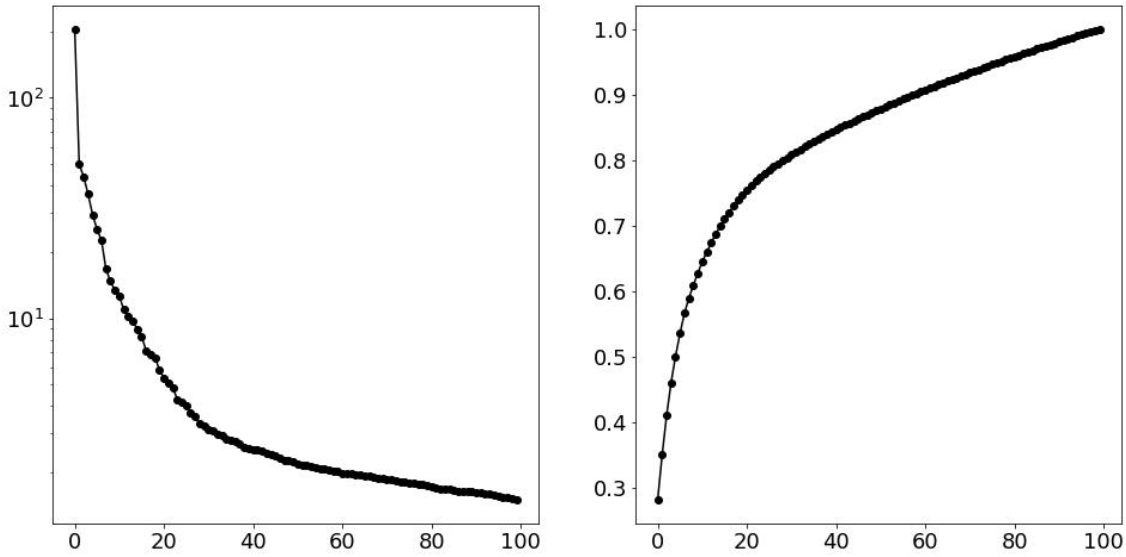


Figure 2.8: First K of eigenvalues and

First plot is plotting the eigen values in Σ. The second plot is result of dividing cumulative sum and the sum of eigen values in Σ, it represents how much variance is captured by first k eigen values

$$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{j=1}^{r} \sigma_j^2}$$

The first plot is plotting the eigenvalues in Σ. The second We can see in the first plot. Only about the first 25 eigenvalues is considerable large. After 50 eigenvalues, the value is slowly decreasing. This means only the first 25 eigenvalues contain a big variance of the dataset, capturing lots of information about our dataset. In the second plot, dividing cumulative sum and the sum of eigenvalues of the first 2 eigenvalues is more than 0.6 - we know that the first 2 eigenvalues capture more than 60% the information of the dataset.

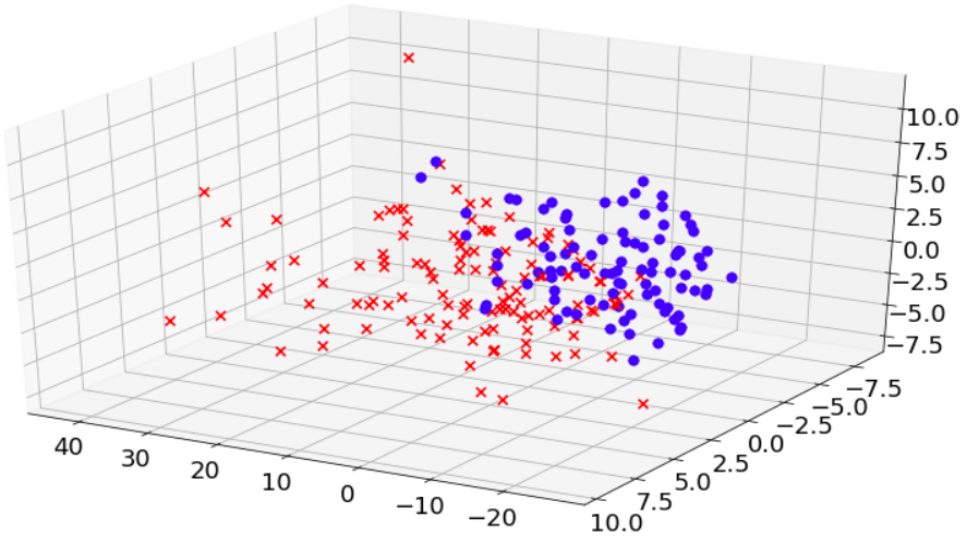


Figure 2.9: Plotting 216 patients in first 3 principal components

In this plot, red x marks represent cancer patients, and blue dots represent non-cancer patients. We can see cancer patients cluster in an area on the left side, the same with non-cancer patients, but they cluster on the right side. So our cancer or non-cancer patients have a relation in the gene data.

Chapter 3

Conclusion

After diving into SVD and PCA, we know that these techniques are powerful for analyzing, reducing high dimensional data. And we also can use these techniques for classifying and compressing data. Besides the advantages of these techniques, it does have drawbacks: information loss, becoming less interpretable. In general, Principal Components Analysis and Singular Value Decomposition is great to use in any situation when working which big data.

Bibliography

[1] Machine Learning Co Ban - Tiep Huu Vu.
Machine Learning Co Ban Ebook: `https://github.com/tiepvupsu/ebookMLCB`

[2] Data-driven Science and Engineering: Machine Learning, Dynamical System, and Controls - Steven L. Brunton, J. Nathan Kutz.
Website and Ebook: `http://databookuw.com/`
Data Github: `https://github.com/dynamicslab/databook_python`

[3] SVD Chapter - Princeton Computer Science Course.
https://www.cs.princeton.edu/courses/archive/spring12/cos598C/svdchapter.pdf

[4] Power Method: `https://www.cs.huji.ac.il/ csip/tirgul2.pdf`

[5] Gram-Schmidt:
https://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/least-squares-determinants-and-eigenvalues/orthogonal-matrices-and-gram-schmidt/MIT18_06SCF11_Ses2.4sum.pdf

[6] Demonstrate in Google Colab:
https://colab.research.google.com/drive/1nTS87DJv-XmnvqKOX$_njJ1XhSN6RoVhQ$