# 9

# 高维数据下的优化算法观点，从 SVD 到 PCA 到 LSA

奇异值分解 *SVD* 和主成分分析 *PCA* 是有内部联系的。我们在这里讨论低秩逼近，并且介绍了一个基于图像处理的有趣案例。

– Jingbo Xia

## 9.1 Intro of SVD

**Theorem 8 ( Singular Value Decomposition)** *For an $m \times n$ matrix $A$ of rank $r$, there exists a factorization (Singular Value Decomposition = SVD) as follows:*

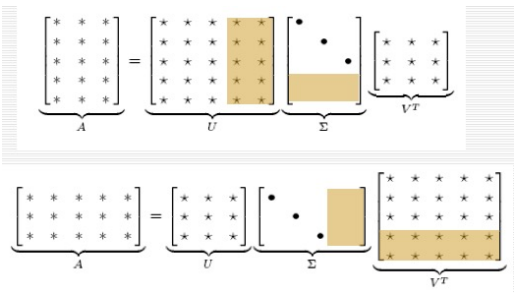
$$A = U\Sigma V^T,$$

*where $U$ and $V$ are orthogonal matrices, $\Sigma = \begin{pmatrix} \Delta & 0 \\ 0 & 0 \end{pmatrix}$, $\Delta = diag(\sigma_1, \cdots, \sigma_r)$, $\sigma_i = \sqrt{\lambda_i}$ (which is called as singular value of $A$), $i = 1, 2, \cdots, r$, $r = Rank(A)$, $\lambda_i$ is the eigenvalue of $AA^T$.*

SVD algorithm is a famous theorem, widely used in image compression, matrix completion, recommendation system, and latent semantic index (sometimes called as latent semantic analysis).

In this section, we will show you a nice proof of SVD theorem.

For better understanding, a $5 \times 3$ real matrix $A$ is shown here:




Here, we rewrite the above formula to

$$A = (A_1, A_2, A_3) = (U_1, \cdots, U_5) \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \\ V_3^T \end{pmatrix},$$

where $A_i$ is the $i$-th column of $A$, $U_k$ is the $k$-th column of $V$, and $V_j$ is the $j$-th column of $V$.

Straightforward computation leads to

$$A = \sigma_1 U_1 V_1^T + \sigma_2 U_2 V_2^T + \sigma_3 U_3 V_3^T.$$

Thus $\sigma_1 U_1 V_1^T + \sigma_2 U_2 V_2^T$ is named as a "low-rank approximation of " $A$. Generally, $\sigma_{i=1}^{R} U_i V_i^T$ is a rank-$R$ approximation of an arbitrary $A$.

**Why low rank approximation**

Why does it need low rank approximation? The answer is to remove sparsity in a proper way. Please think about the case of JD.com for items recommendation and that of drug-receptor matrix for drug recommendation.
Please think it over.

## 9.2 Proof of SVD

**Lemma 1.** Lemma 1:
Suppose $A \in \mathbb{R}^{m \times n}$, the eigenvalues of $A^T A$ and $AA^T$ are nonnegative.

Proof: Suppose $\lambda$ is the eigenvalue of $A^T A$, and $x$ is the corresponding eigenvector, then we have

$$A^T A x = \lambda x.$$

Since $A^T A$ is symmetrical, so $\lambda$ is a real number, and we have

$$0 \leq (Ax, Ax) = (Ax)^T (Ax) = \lambda x^T x.$$

Because $x^T x > 0$, we have $\lambda \geq 0$.

Similarly, we know that eigenvalues of $AA^T$ is nonnegative. □

**Lemma 2.** Lemma 2:
Suppose $A \in \mathbb{R}^{m \times n}$, we have $r(A) = r(A^T A) = r(AA^T)$.

Proof of this lemma is straightforward. Please consider the solution of linear equation system $Ax = 0$, and $A^T Ax = 0$, and proof that the solutions are the same.

**Lemma 3.** Lemma 3:
Suppose $A \in \mathbb{R}^{m \times n}$, then $AA^T$ and $A^T A$ have the same eigenvalues.

Proof. Assume $\lambda$ is an eigenvalue of $A^T A$, then we have $A^T Ax = \lambda x$. Multiplied with $A$ from left side, the equation turns to be $AA^T Ax = \lambda Ax$, and it means that $\lambda$ is as well eigenvalue of $AA^T$.

Similarly, eigenvalue of $AA^T$ is also eigenvalue of $A^T A$. The above two statements suffice to show the lemma.

**Lemma 4.** Lemma 4:
If two matrices are orthogonally equivalent to each other, they have the same singular values.

Proof. Suppose $A, B \in \mathbb{R}^{m \times n}$ are orthogonally equivalent, there exists orthogonal matrix $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$, s.t, $A = UBV$.

From $A^T A = (UBV)^T (UBV) = V^T B^T BV = V^{-1} B^T BV$, we know that $A^T A$ and $B^T B$ have the same eigenvalues, so do the singular values. □

**Main proof**

Proof. Since $A^TA$ is symmetrical matrix, there exists orthogonal matrix $V$, $(r(V) = r)$, s.t.,

$$V^T(A^TA)V = \begin{pmatrix} \Delta^2 & 0 \\ 0 & 0 \end{pmatrix},$$

here $\Delta = diag(\sigma_1, \cdots, \sigma_r)$, $\sigma_i^2 = \lambda_i$.

Splitting $V$, we have

$$V = (V_1, V_2),$$

where $V_1$ consists of the first $r$ columns of $V$.

Note: the split of $V$ is a trick here.

$$\begin{pmatrix} \Delta^2 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} (A^TA)(V_1, V_2)$$

$$= \begin{pmatrix} V_1^TA^TAV_1 & * \\ * & V_2^TA^TAV_2 \end{pmatrix}.$$

Compare the left and right sides, we have

$$V_2^TA^TAV_2 = (AV_2)^T(AV_2) = 0,$$

which shows that $AV_2 = 0$. We also have

$$V_1^TA^TAV_1 = (AV_1)^T(AV_1) = \Delta^2.$$

We denote $U_1 = AV_1\Delta^{-1}$, it is easy to verify that

$$U_1^TU_1 = \Delta^{-1}V_1^TA^TAV_1\Delta^{-1} = E_r,$$

so the first $r$ columns of $U$ (They actually form $U_1$) are orthogonal identical vectors. Hence, there exists an extension of $U_1$ by $U_2 \in \mathcal{R}^{m\times m-r}$ and make

$$U = (U_1, U_2)$$

an orthogonal matrix. Thus we obtain both $V$, $\Sigma$ and $U$.

Therefore,

$$U^TAV = \begin{pmatrix} U_1^T \\ U_2^T \end{pmatrix} A(V_1, V_2) = \begin{pmatrix} U_1^TAV_1 & U_1^TAV_2 \\ U_2^TAV_1 & U_2^TAV_2 \end{pmatrix}$$

$$= \begin{pmatrix} U_1^TU_1\Delta & U_1^T(AV_2) \\ U_2^TU_1\Delta & U_2^T(AV_2) \end{pmatrix} = \begin{pmatrix} \Delta & 0 \\ 0 & 0 \end{pmatrix},$$

as both $U_2^TU_1$ and $AV_2$ equal to 0.

So the theorem follows.

## 9.3 Conclusion of SVD

1. Exercise.

   Please do not forget the subsequent exercise in this afternoon on the blackboard, where $A$ is $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$.

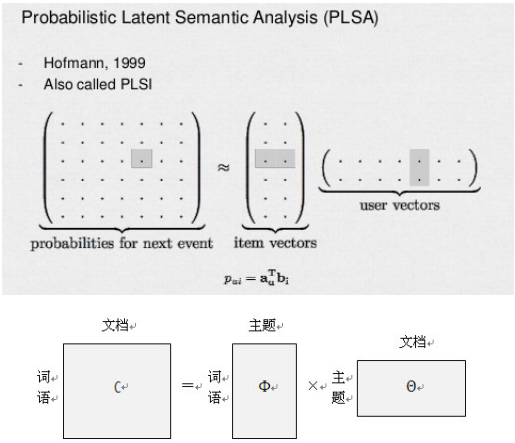
   Answer sheet:

$$A = \begin{pmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sqrt{3} & 0 \end{pmatrix} \begin{pmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix}.$$

2. Please keep in mind the image compression example, and also the recommendation system example of TaoBao purchase.
3. SVD, along with matrix decomposition, is the beginning of tensor decomposition discussion.
4. A question? What is and Why does low-rank approximation needed in SVD?

## 9.4 Application to Latent Semantic Analysis

Here are showing how SVD working for LSA.




**Why low-rank approximation?**

SVD in image compression might be a way to explain the reason for low-rank approximation, and in the meantime, Eckart-Young-Mirsky Theorem evaluates the approximation via estimating the Frobenius norm of $||A - \tilde{A}||_F^2$

**Theorem 9 (Eckart-Young-Mirsky Theorem)** *For an $m \times n$ matrix $A$ of rank $r$, assume the SVD is $A = U\Sigma V^T$, where $\Sigma = \begin{pmatrix} \Delta & 0 \\ 0 & 0 \end{pmatrix}$, $\Delta = diag(\sigma_1, \cdots, \sigma_r)$. For an approximation of $A$ as $\tilde{A} = U\tilde{\Delta}V^T$, where $\tilde{\Delta} = diag(\sigma_1, \cdots, \sigma_l)$, we have*

$$\min_{rank(\tilde{A} \leq l)} ||A - \tilde{A}||_F = \sigma_l^2 + \cdots + \sigma_r^2.$$

## 9.5 Intro to PCA

Principal component analysis (PCA) aims to use "an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called **principal component**"[1].

As shown in figure 9.2, different selection of projection line leads to difference in the variance. It's no hard to observe that the left projection brings greater variance than the right projection does.

Then, how to calculate the projection coordinates?

Let's discuss it in a $R^n$ space. If $\alpha$ project to $\beta$, then the projection[2] is

$$\frac{(\alpha, \beta)}{(\beta, \beta)}\beta = \frac{(\alpha, \beta)}{|\beta|}\frac{\beta}{|\beta|},$$

[1] https://en.wikipedia.org/wiki/Principal_component_analysis

[2] Note: This is a vector, and this vector is with the same/ opposite direction of $\beta$

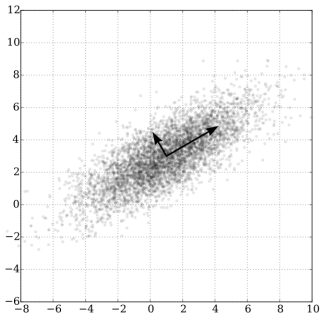


Figure 9.1: PCA of a multivariate Gaussian distribution centered at $(1, 3)$ with a standard deviation of 3 in roughly the $(0.866, 0.5)$ direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

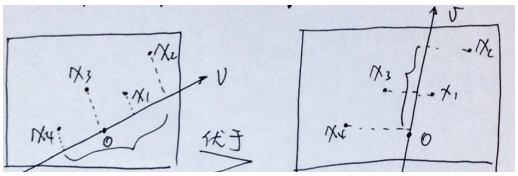


Figure 9.2: An example of four samples/points which are projected to different vector/direction.

while the projection coordinate[3] in $\beta$ is

$$\frac{(\alpha, \beta)}{|\beta|}.$$

Without the loss of the generality, we assume $|\beta| = 1$, and then the projection coordinate is $(\alpha, \beta)$.

Now it is time to discuss the variance. Assume there are $l$ samples $X_i \in \mathbb{R}^d$, $(i = 1, 2, \cdots, l)$, and $V$ is a vector to which $X_i$ projects. For simplicity, we assume that the mean of samples $\bar{X} = 0$. Denote $\sigma^2$ as variance of all of the projection coordinates, we know

$$\begin{aligned} \sigma^2 &= \frac{1}{l}\sum_{i=1}^{l}(V^T X_i - 0)^2 = \frac{1}{l}\sum_{i=1}^{l}(V^T X_i)(V^T X_i)^T \\ &= \frac{1}{l}\sum_{i=1}^{l} V^T X_i X_i^T V = V^T(\frac{1}{l}\sum_{i=1}^{l} X_i X_i^T)V \\ &:= V^T C V \end{aligned} \tag{9.1}$$

We denote $C$ as the covariance matrix.

As a general form, i.e., if we remove the assumption that the mean of $X_i$ equals to zero, the above deduction leads to:

$$C = \frac{1}{l}\sum_{i=1}^{l}(X_i - \bar{X})(X_i - \bar{X})^T.$$

**Solve of PCA by optimization theory**

The optimization problem with PCA is

$$V = \underset{V \in \mathbb{R}^d, |V|=1}{\arg\max}\ V^T C V \tag{9.2}$$

The Lagrangean function is

$$f(v, \lambda) = V^T C V - \lambda(V^T V - 1).$$

[3]Note: This is a scalar value.

By solving $\frac{\partial f}{\partial V} = 0$, we have

$$CV = \lambda V.$$

This means that $V$, as a projection vector, is the eigenvector of the covariance matrix.

By sorting the eigenvalue with a descending order, one can select different $V_j$, and project $X_i \in \mathbb{R}^d$ to different $V_j$ and obtain corresponding projection coordinate $(X_i, V_j) = X_i^T V_j$.

[**Feature reduction**] If we fixed the first $J$ $V_j$ with greatest eigenvalues, and replace $X_i$ with a $J$-tuple vector, $(X_i^T V_1, X_i^T V_2, \cdots, X_i^T V_J)$, we convert the $d$-tuple vector $X_i$ into a $J$-tuple one. This is feature reduction.

[**Data reconstruction**] If we represented the sample $X_i$ with an approximation, $(X_i^T V_1)V_1 + (X_i^T V_2)V_2 + \cdots + (X_i^T V_J)V_J$, this is a low-rank approximate of $X_i$. We call it data reconstruction.

## 9.6 SVD and PCA

SVD and principle component analysis have close relevance.

Assume the matrix $A \in \mathbb{R}^{m \times n}$ contains $m$ "$n$-tuple" sample vectors $A_i$ $(i = 1, 2, .., \cdots, m)$, i.e.,

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{pmatrix}, \tag{9.3}$$

The purpose of PCA is to find several "$n$-tuple" principle components $V_1^T, \cdots, V_r^T$, (here $V_j$ $(j = 1, 2, \cdots, r)$ are by chance the eigenvectors of co-variance matrix $A^T A$,) s.t.,

$$A_i = b_{i1} V_1^T + b_{i2} V_2 + \cdots + b_{ir} V_r^T,$$

in another word,

$$A = B V^T.$$

Amazingly, SVD solution of $A = U \Sigma V^T$ is a great co-incidence here!

Note: For better presentation, I put the sample vector $A_i$ in a horizontal way, and eigenvector $V_j$ in a vertical way.

**Two questions**

Could I suggest you to figure out why we treat $V_j$ as the eigenvector of $A^T A$ in PCA?
And how in details is $A^T A$ treated as a co-variance matrix in PCA?
Please think it over.

Generally speaking, it is never easy to figure out how $A_i$ and $V_j$ look like. With a nice visualization material, an easily-understood blog is strongly suggested here, `http://www.infoq.com/cn/articles/matrix-decomposition-of-recomm`

In this example, $A_i$ is a 4096-tuple vector which presents a picture, and $V_j$s is so called "creepy faces" which represent different characteristics of faces and weighted sum of "creepy faces" reconstruct $A_I$. WATCH, and ENJOY!

$A_i$, each one will be approximated by a weighted sum of the below "creepy faces":

$V_j^T$, "creepy faces":

Python Codes

```
from sklearn.datasets import fetch_olivetti_faces
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt
%matplotlib inline
faces = fetch_olivetti_faces()
print(faces.DESCR)
```

There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

Python Codes

```
# Here are the first ten guys of the dataset
fig = plt.figure(figsize=(10, 10))
for i in range(5):
    ax = plt.subplot2grid((1, 5), (0, i))
    ax.imshow(faces.data[i * 10].reshape(64, 64), cmap=plt.cm.gray)
    ax.axis('off')

# Let's compute the PCA
pca = PCA()
pca.fit(faces.data)
```

Python Codes

```
# Now, the creepy guys are in the components_ attribute.
# Here are the first four ones:

fig = plt.figure(figsize=(10, 10))
for i in range(4):
    ax = plt.subplot2grid((2, 2), (0, i))
```

```
    ax.imshow(pca.components_[i].reshape(64, 64), cmap=plt.cm.gray)
    ax.axis('off')
```

Python Codes

```
# Reconstruction process

from skimage.io import imsave
face = faces.data[0]  # we will reconstruct the first face

# During the reconstruction process we are actually computing,
# at the kth frame, a rank k approximation of the face.
# To get a rank k approximation of a face,
# we need to first transform it into the 'latent space', and then
# transform it back to the original space
# Step 1: transform the face into the latent space.
# It's now a vector with 400 components. The kth component gives
# the importance of the kth  creepy guy

trans = pca.transform(face.reshape(1, -1))
# Reshape for scikit learn
```

Python Codes

```
# Step 2: reconstruction. To build the kth frame, we use all the creepy guys
# up until the kth one.
# Warning: this will save 400 png images.

for k in range(400):
    rank_k_approx = trans[:, :k].dot(pca.components_[:k])
            + pca.mean_
    imsave('{:>03}'.format(str(k))
            + '.jpg', rank_k_approx.reshape(64, 64))
```