Introduction

- Principal component analysis involves **extracting** linear composites of observed variables. So PCA is actually a feature extraction technique. Along the principal axis of the new coordinate system, the variance is maximized.
- PCA uses 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 components, which are uncorrelated. **Comments:** Note the no-correlation means no linearly uncorrelated. But beyond linear, it might still be correlated.
- A tentative intuition: The looking for maximum variance is also related to the eigenvectors and eigenvalues of the covariance matrix. In a diagonalized matrix, all the former off-diagonal 'covariances' can be understood as 'collapsing' to the diagonal elements, which are 'variances'. Maximum variance is obtained when the matrix is diagonalized, that is, when the covariance matrix is in the basis formed by its eigenvectors. However, this is still an intuitive picture. Next will provide **more details on how eigenvectors of the covariance matrix will retain maximum information**.

Derivation of PCA

An intuition before the derivation

- We start with a random vector $\mathbf{x}$, with its samples evaluated in a standard basis. Suppose this random vector is a 2D random vector, then its samples is just a cloud of dots described by 2D vectors in the 2D xy plane. Now we project the cloud of data to the x axis and obtain many points along the axis. We then calculate its variance using the values on the x axis.
- We can also do the similar thing by projecting the data to the y axis and calculate the variance of the data along the y axis.
- We currently obtain two variance values respectively obtained from projecting the same vectors from random vector $\mathbf{x}$. These two values should normally different. We can also project sample vectors to any direction rather than along x or y. Now the questions, along which direction (or which vector) can we obtain a maximum variance?

Formal derivation

- We now go beyond the 2D random vector, but consider any higher dimensional random vector $\mathbf{x}$. We want to find a direction to project the random vector samples and simultaneously we obtain a maximum variance along this direction.
- We denote this direction as a vector u with $||u||_2^2 = 1$. The projection of the random vector $\mathbf{x}$ on u is thus $\mathbf{x} \cdot u$. Note this is not projecting $\mathbf{x}$ to a basis of vectors but just one vector u. Or we may say we project $\mathbf{x}$ onto a single basis vector u.

- Because u is a normal vector (as compared to a random vector), so the projection $\mathbf{x} \cdot u$ is just the linear expansion of the scalar random variables in $\mathbf{x}$. Therefore, $\mathbf{x} \cdot u$ is just a scalar random variable and we may calculate its variance.
- The variance of sample dots projected onto u thus can be calculated as
$$E[(\mathbf{x} \cdot u - E[\mathbf{x} \cdot u])^2] = E[(u \cdot (\mathbf{x} - E[\mathbf{x}]))^2] = u^T E[(\mathbf{x} - E[\mathbf{x}]) \cdot (\mathbf{x} - E[\mathbf{x}])^T] u = u^T C u$$

Comments:

- In the future, the form of $u^T C u$ is no longer a bizarre quantity. At least, if C is a symmetric metric, this quantity can be understood as the **variance** of samples projected onto a vector u.
- Understand how the positive definite of a matrix is also related to the form $u^T C u$. Because $u C^T u$ corresponds to variance when C is symmetric, and variance is always bigger or equal to zero, so symmetric matrix C is always at least semi positive definite.

Find the u that maximizes the variance

- Now the problem becomes
$$\max_{u \in \mathbb{R}^n} u^T C u \text{ subject to } \|u\|_2^2 = 1$$
If without the condition $\|u\|_2^2 = 1$, then the variance can be any number as u scales, https://stats.stackexchange.com/questions/117695/why-is-the-eigenvector-in-pca-taken-to-be-unit-norm (https://stats.stackexchange.com/questions/117695/why-is-the-eigenvector-in-pca-taken-to-be-unit-norm)
- The standard way of solving the above optimization problem with equality constraints is by forming the Lagrangian, which in this case is given by
$$L(u, \lambda) = u^T C u - \lambda u^T u$$
where λ is the Lagrangian multiplier associated with the equality constraints.
- By taking derivative w.r.t. u and setting it to zero, we have
$$\nabla_u L(u, \lambda) = \nabla_u (u^T C u - \lambda u^T u) = 2C^T u - 2\lambda u = 0$$
- Matrix C is symmetric due to the specific definition above, thus we have
$$Cu = \lambda u$$
This shows that the only vector u that can maximize the variance is the eigenvector of C with biggest eigenvalue.
- Because the covariance matrix C is symmetric, the eigenvectors of C are orthogonal. Therefore, PCA the components found are **uncorrelated** to each other.

Why do PCA?

- In the previous cell, we derive the PCA by maximizing the variance. However, maximizing the variance is not the original purpose of PCA.

- From above derivation, we actually transform the data vector $\mathbf{x}_i$ (denoting a sample from random vector $\mathbf{x}$) from standard basis to another orthogonal (unitary) basis with basis vectors $u_1, u_2, \ldots$. We just find one of u_i's that makes the variance is maximum.
- Because unitary transformation preserves the norm, so if we calculate the norm in both bases, then the result should be same. In other words, no error with whole basis used.
- The purpose of PCA is to use as few as possible the basis vectors u_i to represent the original data. There must be errors if we don't use the whole basis of $\{u_i\}$. However, we find that minimizing such an error is just equivalent to maximizing the variance. https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch18.pdf (https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch18.pdf)
- Therefore, if we want to use PCA and employ small number of basis vectors to represent the original data, we need make sure that the variance on the subset of u_i should contain most of variance in all u_i.

Are there implicit Gaussian assumptions in the use of PCA?

https://www.quora.com/Are-there-implicit-Gaussian-assumptions-in-the-use-of-PCA-principal-components-analysis (https://www.quora.com/Are-there-implicit-Gaussian-assumptions-in-the-use-of-PCA-principal-components-analysis)
For multivariate gaussian distributions, zero correlation between components implies independence, a property that is not true for most distributions. Another way of thinking about this is that gaussian distributed data have no higher order statistics beyond variance; therefore PCA is extremely efficient at representing multivariate normally distributed data.

Many people do not agree that PCA make implicit Gaussian assumption.

Application of PCA

Examples of benefiting from using PCA

- Represent the original data within this low-dimensional subspace: data compression.
- Perform data visualization if dimensions of the subspace is low enough.
- Image recognition: We can also efficiently calculate the distance between the projections of the original data on a few major principal axes. By comparing the distances, we can do image recognition. The key should be: in the reduced dimension, the distance is much easier to calculate. See https://en.wikipedia.org/wiki/Eigenface#Use_in_facial_recognition (https://en.wikipedia.org/wiki/Eigenface#Use_in_facial_recognition) for an introduction. Be familiar with the geometric picture of the subtraction of two vectors. The different vector goes from the end of one vector to the other.
- Do anomaly detection by calculating whether the distance of a data to the principal axis is very different from others. This is different from the image or pattern recognition. However, here we still make use of the advantage of using PCA, which is to calculate the distance very efficiently in a reduced dimension. Always use the 2D correlated data for intuitive picture. In 2D case, we may calculate

distance just from 1D.
- Do dimensional reduction in supervised learning.
- Latent semantic indexing (LSI). We find similar articles by the similar approach as image recognition: comparing distances projected to principal axes. In LSI, usually skip the normalization because this will amplify the infrequent words. The similarity between articles is define as the $\cos\theta = \frac{x^{(i)T}x^{(j)}}{\|x^{(i)}\|^2\|x^{(j)}\|^2}$. Question: If we can compare distance to find similar articles, what is the purpose of using similarity defined above?

Examples when PCA fails

- PCA find new variables that maximize the variance. This will cause problems in some classification tasks where the feature points of different classes lie almost in parallel to the principal axis. In such a case, the projection of these data from different classes will be mixed and thus hard to separate. To handle such problems, LDA might be a good choice.
- In the cases where ICA performs very well (elaborate later).

PCA implementations

Technicals of PCA

- There are two conventions for storing a data in matrix. One is more like a flat table where n and p in a $n \times p$ matrix X describe the number of samples and the number of variables respectively. In this case, covariance matrix is $C = \frac{X^TX}{n-1}$. Also this form is only true **when the data is centered**. The other way use n and p to describe the number of variables and the number of samples. The covariance matrix will be $C = \frac{XX^T}{p-1}$. This description is more consistent with the linear algebra where column is used for a vector.
- The X matrix is already centered and thus the covariance matrix can be written as X^TX or XX^T. Otherwise we have to explicitly subtract the mean. Therefore, in practice, whether centering data (and normalization) is very sensitive in the calculation, depending how the software package handle the calculation of covariance.
- Although PCA is typically introduced by diagonalizing the covariance matrix, people usually avoid doing so due to the numerical instability when forming the covariance matrix (see notes about linear algebra). Instead, people often implement PCA by using SVD.
- If we are mainly do dimension reduction, then we can use SVD to replace PCA, as below:

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

$$A = U_r\Sigma_r V_r^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

Confusion naming issues about PCA

- Here we use the covariance matrix with the version X^TX with matrix V formed by the eigenvectors of covariance matrix. These eigenvectors in V are called **principal axes** or **principal directions** of the data.
- Projections of the data on the principal axes are called **principal components** or **PC scores**. Note however, people sometimes use these concepts for slightly different meanings.
- The jth principal component is given by jth column of XV. The coordinates of the ith data point in the new PC space are given by the ith row of XV. This form is correct for the case that the row of X is for a data point (a vector). The first row of X dot multiply each column of V (projections) gives the first row of XV matrix. This row gives the projections of first data point (vector) (the first row of X) on different V vectors(eigenvectors). Similarly, the first column of XV gives the projections of all the data points (all the rows of X) on the first principal axis (the first column of V).
- Once we understand the fundamentals, it is straightforward to figure out the form of principal components in the case of covariance with form of XX^T. Also note it is not necessarily use column to describe the principal component as in the case of XV. This depends on how we arrange the multiplication of the matrix.
- Some notation confusion clarified in the link: Even though v_1 (one of the eigenvectors of the covariance matrix, e.g. the first one) and Xv_1 (projection of the data onto the 1-dimensional subspace spanned by v_1) are two different things, both of them are often called "principal component", sometimes even in the same text.

Mapping PCA terms to SVD

https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca (https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca)

- Assuming SVD of $X = USV^T$, then it is easy to show $C = V\frac{S^2}{n-1}V^T$. This means that right singular vectors V are principal directions and that singular values are related to the eigenvalues of covariance matrix via $\lambda_i = \frac{s_i^2}{n-1}$. Note eigenvalues λ_i show variances of the respective PCs. Principal components (scores) are given by $XV = USV^TV = US$.
- Standardized score are given by columns of $\sqrt{n-1}U$ and loadings are given by columns of $\frac{VS}{n-1}$. See https://stats.stackexchange.com/questions/125684/how-does-fundamental-theorem-of-factor-analysis-apply-to-pca-or-how-are-pca-l (https://stats.stackexchange.com/questions/125684/how-does-fundamental-theorem-of-factor-analysis-apply-to-pca-or-how-are-pca-l) and https://stats.stackexchange.com/questions/143905/loadings-vs-eigenvectors-in-pca-when-to-use-one-or-another (https://stats.stackexchange.com/questions/143905/loadings-vs-eigenvectors-in-pca-when-to-use-one-or-another) for why 'loadings' should not be confused with principal directions.
- **The above is correct only** if X is centered and only the covariance matrix is equal to $X^TX/(n-1)$.
- **The above is correct only** for X having samples in rows and variables in columns. Otherwise, U and V exchange interpretations.

- If one wants to perform PCA on a correlation matrix (instead of a covariance matrix), then columns of X should not only be centered, but standardized as well, i.e., divided by their standard deviations.
- To reduce the dimensionality of the data from p to $k < p$, select k first columns of U, and $k \times k$ upper-left part of S. Their product $U_k S_k$ is the required $n \times k$ matrix containing first k PCs.
- Further multiplying the first k PCs by the corresponding principal axes V_k^T yields $X_k = U_k S_k V_k^T$ matrix that has the original $n \times p$ size but is of lower rank (of rank k). This matrix X_k provides a reconstruction of the original data from the first k PCs. It has the lowest possible reconstruction error, as shown in https://stats.stackexchange.com/questions/130721/what-norm-of-the-reconstruction-error-is-minimized-by-the-low-rank-approximation (https://stats.stackexchange.com/questions/130721/what-norm-of-the-reconstruction-error-is-minimized-by-the-low-rank-approximation).
- Strictly speaking, US is of $n \times n$ size and V is of $p \times p$ size. However, if $n > p$ then the last $n - p$ columns of U are arbitrary (and corresponding rows of S are zero); one should therefore use an economy size (or thin) SVD that returns U of $n \times p$ size, dropping the useless columns. For large $n >> p$ the matrix U would otherwise be unnecessarily huge. The same applies for an opposite situation of $n << p$.

Implement PCA by diagonalizing covariance matrix and by SVD

https://math.stackexchange.com/questions/2359992/how-to-resolve-the-sign-issue-in-a-svd-problem (https://math.stackexchange.com/questions/2359992/how-to-resolve-the-sign-issue-in-a-svd-problem)
https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca (https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca)

In [2]:

```
import numpy as np
from numpy import linalg as la
np.random.seed(42)

def flip_signs(A, B):
    """
    utility function for resolving the sign ambiguity in SVD
    http://stats.stackexchange.com/q/34396/115202
    """
    signs = np.sign(A) * np.sign(B)
    #print(B*signs)
    return A, B * signs
    #Since the normalized eigenvectors are unique only up to a sign,eigenvectors calcuated
    #with different approaches might have a sign difference.
    #The function here will flip the sign of one vector if the signs of two vectors are different.


# Let the data matrix X be of n x p size, where n is the number of samples and p is
# the number of variables
n, p = 5, 3
X = np.random.rand(n, p) #generate a nxp matrix with random numbers.

# Centerizing the data
X -= np.mean(X, axis=0) # X = X - np.mean(X, axis=0)
# A simple way to center the data

# the p x p covariance matrix
C = np.cov(X, rowvar=False)
# Be careful the use of rowvar = False. Here each row is not a variable, but many variables.
print ("C = \n", C)

# C is a symmetric matrix and so it can be diagonalized:
l, principal_axes = la.eig(C)

# sort results wrt. eigenvalues. This indicates that the 'raw' results are not ordered? Always?
#Returns the indices that would sort this array.
idx = l.argsort()[::-1]

l, principal_axes = l[idx], principal_axes[:, idx]

# the eigenvalues in decreasing order
print( "l = \n", l)
```

```
# a matrix of eigenvectors (each column is an eigenvector)
print( "V = \n", principal_axes)

# projections of X on the principal axes are called principal components.
# Figure this out by checking .dot() elsewhere.
principal_components = X.dot(principal_axes)
print ("Y = \n", principal_components)

# we now perform singular value decomposition of X
# "economy size" (or "thin") SVD.
U, s, Vt = la.svd(X, full_matrices=False) # Shift+Tab see the meaning of full_matrices = False.
V = Vt.T
S = np.diag(s)



# 1) then columns of V are principal directions/axes.
assert np.allclose(*flip_signs(V, principal_axes))

# 2) columns of US are principal components
assert np.allclose(*flip_signs(U.dot(S), principal_components))

# 3) singular values are related to the eigenvalues of covariance matrix
assert np.allclose((s ** 2) / (n - 1), l)

# 8) dimensionality reduction
k = 2
PC_k = principal_components[:, 0:k]
US_k = U[:, 0:k].dot(S[0:k, 0:k])
assert np.allclose(*flip_signs(PC_k, US_k))

# 10) we used "economy size" (or "thin") SVD
assert U.shape == (n, p)
assert S.shape == (p, p)
assert V.shape == (p, p)
```

```
C =
 [[ 0.09338628 -0.11086559 -0.02943783]
 [-0.11086559  0.18770817  0.0336127 ]
 [-0.02943783  0.0336127   0.12511719]]
l =
 [0.27418905 0.11232653 0.01969604]
V
```

```
v =
 [[ 0.53435576  0.10510519 -0.83869948]
 [-0.79577968 -0.27194755 -0.54109078]
 [-0.28495372  0.95655498 -0.06167616]]
Y =
 [[-0.5382821   0.04170504 -0.17101639]
 [ 0.37801268 -0.26959854  0.10654358]
 [-0.60281427 -0.09375913  0.14821045]
 [ 0.31232627  0.5572872   0.03786103]
 [ 0.45075742 -0.23563458 -0.12159868]]
(5, 3)
```