PCA, K-Means, Gaussian Mixture

Milan Straka

📅 December 14, 2020
Unsupervised Machine Learning
The principal component analysis, PCA, is a technique used for

- dimensionality reduction,
- feature extraction,
- whitening,
- data visualization.

To motivate the dimensionality reduction, consider a dataset consisting of a randomly translated and rotated image.

![Figure 12.1 of Pattern Recognition and Machine Learning.](image)

Every member of the dataset can be described just by three quantities – horizontal and vertical offsets and a rotation. We usually say that the data lie on a manifold of dimension three.
We start by defining the PCA in two ways.

**Maximum Variance Formulation**

Given data \( \mathbf{x}_1, \ldots, \mathbf{x}_N \) with \( \mathbf{x}_i \in \mathbb{R}^D \), the goal is to project them to a space with dimensionality \( M < D \), so that the variance of their projection is maximal.

We start by considering a projection to one-dimensional space. Such a projection is defined by a vector \( \mathbf{u}_1 \), and because only the direction of \( \mathbf{u}_1 \) matters, we assume that \( \mathbf{u}_1^T \mathbf{u}_1 = 1 \).

We start by pointing out that the projection of \( \mathbf{x}_i \) to \( \mathbf{u}_1 \) is given by \( (\mathbf{u}_1^T \mathbf{x}_i) \mathbf{u}_1 \), because the vectors \( \mathbf{u}_1 \) and \( \mathbf{x}_i - (\mathbf{u}_1^T \mathbf{x}_i) \mathbf{u}_1 \) are orthogonal:

\[
\mathbf{u}_1^T (\mathbf{x}_i - (\mathbf{u}_1^T \mathbf{x}_i) \mathbf{u}_1) = \mathbf{u}_1^T \mathbf{x}_i - (\mathbf{u}_1^T \mathbf{x}_i) \mathbf{u}_1^T \mathbf{u}_1 = 0.
\]
Principal Component Analysis

We therefore use $u_1^T x_i$ as the projection of $x_i$. If we define $\bar{x} = \sum_i x_i / N$, the mean of the projected data is $u_1^T \bar{x}$ and the variance is given by

$$\frac{1}{N} \sum_{i=1}^{N} (u_1^T x_i - u_1^T \bar{x})^2 = u_1^T S u_1,$$

where $S$ is the data covariance matrix defined as

$$S = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x}) (x_i - \bar{x})^T.$$

We can write the data covariance matrix in matrix form as $S = \frac{1}{N} (X - \bar{x})^T (X - \bar{x})$. If the original data is centered (it has zero mean), then $S = \frac{1}{N} X^T X$, which we have already encountered.
Principal Component Analysis

To maximize $u_1^T Su_1$, we need to include the constraint $u_1^T u_1$ by introducing a Lagrange multiplier $\lambda_1$ for the constraint $u_1^T u_1 - 1 = 0$ and then maximizing the Lagrangian

$$L = u_1^T Su_1 - \lambda_1 (u_1^T u_1 - 1).$$

By computing a derivative with respect to $u_1$, we get

$$Su_1 = \lambda_1 u_1.$$

Therefore, $u_1$ must be an eigenvector of $S$ corresponding to eigenvalue $\lambda_1$.

Because the value to maximize $u_1^T Su_1$ is then $u_1^T \lambda_1 u_1 = \lambda_1 u_1^T u_1 = \lambda_1$, the maximum will be attained for eigenvector $u_1$ corresponding to the largest eigenvalue $\lambda_1$.

The eigenvector $u_1$ is known as the **first principal component**.

For a given $M$, the principal components are eigenvectors corresponding to $M$ largest eigenvalues, and maximize the variance of the projected data.
Principal Component Analysis

Minimum Error Formulation

Assume $u_1, \ldots, u_D$ is some orthonormal set of vectors, therefore, $u_i^T u_j = [i == j]$.

Every $x_i$ can be then expressed using this basis as

$$x_i = \sum_j (x_i^T u_j) u_j,$$

using a similar argument as the one we used to derive the orthogonal projection.

Because we want to eventually represent the data using $M$ dimensions, we will approximate the data by the first $M$ basis vectors:

$$\tilde{x}_i = \sum_{j=1}^M z_{i,j} u_j + \sum_{j=M+1}^D b_j u_j.$$
We now choose the vectors $u_j$, coordinates $z_{i,j}$ and biases $b_j$ to minimize the approximation error, which we measure as a loss

$$L = \frac{1}{N} \sum_{i=1}^{N} \left\| x_i - \tilde{x}_i \right\|^2.$$

To minimize the error, we compute the derivative of $L$ with respect to $z_{i,j}$ and $b_j$, obtaining

$$z_{i,j} = x^T_i u_j, \quad b_j = \bar{x}^T u_j.$$

Therefore, we can rewrite the loss as

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=M+1}^{D} (x^T_i u_j - \bar{x}^T u_j)^2 = \sum_{j=M+1}^{D} u_j^T S u_j.$$

Analogously, we can minimize $L$ by choosing eigenvectors of $D - M$ smallest eigenvalues.
PCA Applications – Data Compression

We can represent the data $x_i$ by the approximations $\tilde{x}_i$

$$\tilde{x}_i = \sum_{j=1}^{M} (x_i^T u_j) u_j + \sum_{j=M+1}^{D} (\bar{x}^T u_j) u_j = \bar{x} + \sum_{j=1}^{M} (x_i^T u_j - \bar{x}^T u_j) u_j.$$
PCA Applications – Data Compression

Figure 12.4 of Pattern Recognition and Machine Learning.

Figure 12.3 of Pattern Recognition and Machine Learning.
The PCA formula allows us to perform **whitening** or **sphering**, which is a linear transformation of the given data, so that the resulting dataset has zero mean and identity covariance matrix. Notably, if $U$ are the eigenvectors of $S$ and $\Lambda$ is the diagonal matrix of the corresponding eigenvalues (i.e., $SU = U\Lambda$), we can define the transformed data as

$$z_i \overset{\text{def}}{=} \Lambda^{-1/2}U^T(x_i - \bar{x}).$$

Then, the mean of $z_i$ is zero and the covariance is given by

$$\frac{1}{N} \sum_i z_i z_i^T = \frac{1}{N} \sum_i \Lambda^{-1/2}U^T(x_i - \bar{x})(x_i - \bar{x})^T U \Lambda^{-1/2} = \Lambda^{-1/2} U S U \Lambda^{-1/2} = \Lambda^{-1/2} \Lambda \Lambda^{-1/2} = I.$$
PCA Applications – Whitening or Sphering

Figure 12.6 of Pattern Recognition and Machine Learning.

PCA
PowerIteration
Clustering
KMeans
MultivariateGaussian
GaussianMixture
Note that PCA does not have access to supervised labels, so it may not give a solution favorable for further classification.

Figure 12.7 of Pattern Recognition and Machine Learning.
Note that it can be proven that if we construct a MLP autoencoder, which is a model trying to reconstruct input as close as possible, then even if the hidden layer uses non-linear activation, the solution to a MSE loss is a projection onto the $M$-dimensional subspace defined by the first $M$ principal components (but is not necessary orthonormal or orthogonal).
However, non-linear PCA can be achieved, if both the encoder and the decoder are non-linear.

Figure 12.9 of Pattern Recognition and Machine Learning.
There are two frequently used algorithms for performing PCA.

If we want to compute all (or many) principal components, we can compute directly the eigenvectors and eigenvalues of the covariance matrix.

We can even avoid computing the covariance matrix. If we instead compute the singular value decomposition of $(X - \bar{x}) = UDV^T$, it holds that

$$(X - \bar{x})^T (X - \bar{x}) = V D U^T U D V^T = V D^2 V^T.$$ 

Therefore,

$$(X - \bar{x})^T (X - \bar{x}) V = V D^2,$$

which means that $V$ are the eigenvectors of $(X - \bar{x})^T (X - \bar{x})$ and therefore of the data covariance matrix $S$. The eigenvalues of $S$ are the squares of the singular values of $(X - \bar{x})$ divided by $N$. 

If we want only the first (or several first) principal components, we might use the **power iteration algorithm**.

The power iteration algorithm can be used to find a **dominant** eigenvalue (an eigenvalue with absolute value strictly larger than absolute value of all other eigenvalues) and the corresponding eigenvector (it is used for example to compute PageRank). It works as follows:

**Input**: Diagonalizable matrix $A$ with a dominant eigenvalue.

**Output**: The dominant eigenvalue $\lambda$ and the corresponding eigenvector $v$, with probability close to 1.

- Initialize $v$ randomly (for example each component from $U[-1, 1]$).
- Repeat until convergence (or for a fixed number of iterations):
  - $v \leftarrow Av$
  - $\lambda \leftarrow \|v\|$
  - $v \leftarrow v/\lambda$

If the algorithm converges, then $v = Av/\lambda$, so $v$ is an eigenvector with eigenvalue $\lambda$. 
Computing PCA — The Power Iteration Algorithm

In order to analyze the convergence, let \((\lambda_1, \lambda_2, \lambda_3, \ldots)\) be the eigenvalues of \(A\), in the descending order of absolute values, so \(|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots\), where the strict equality is the consequence of the dominant eigenvalue assumption.

If we express the vector \(v\) in the basis of the eigenvectors as \((a_1, a_2, a_3, \ldots)\), then \(Av/\lambda_1\) is in the basis of the eigenvectors:

\[
\frac{Av}{\lambda_1} = \left( \frac{\lambda_1}{\lambda_1} a_1, \frac{\lambda_2}{\lambda_1} a_2, \frac{\lambda_3}{\lambda_1} a_3, \ldots \right) = \left( a_1, \frac{\lambda_2}{\lambda_1} a_2, \frac{\lambda_3}{\lambda_1} a_3, \ldots \right).
\]

Therefore, all but the first coordinates decreased by at least a factor of \(|\lambda_2/\lambda_1|\).

If the initial \(v\) had a non-zero first coordinate \(a_1\) (which has probability very close to 1), then repeated multiplication with \(A\) will converge to the eigenvector corresponding to \(\lambda_1\).
After we get the largest eigenvalue $\lambda_1$ and its eigenvector $v_1$, we can modify the matrix $A$ to “remove the eigenvalue $\lambda$”. Considering $A - \lambda_1 v_1 v_1^T$:

- multiplying it by $v_1$ returns zero:

$$ (A - \lambda_1 v_1 v_1^T)v_1 = \lambda_1 v_1 - \lambda_1 v_1 v_1^T v_1 = 0, $$

- multiplying it by other eigenvectors $v_i$ gives the same result:

$$ (A - \lambda_1 v_1 v_1^T)v_i = Av_i - \lambda_1 v_1 v_1^T v_i = Av_i. $$
We are now ready to formulate the complete algorithm for computing the PCA.

**Input:** Matrix $X$, desired number of dimensions $M$.

- Compute the mean $\mu$ of the examples (the rows of $X$).
- Compute the covariance matrix $S \leftarrow \frac{1}{N} (X - \mu)^T (X - \mu)$.
- for $i$ in $\{1, 2, \ldots, M\}$:
  - Initialize $v_i$ randomly.
  - Repeat until convergence (or for a fixed number of iterations):
    - $v_i \leftarrow S v_i$
    - $\lambda_i \leftarrow \|v_i\|$
    - $v_i \leftarrow v_i / \lambda_i$
  - $S \leftarrow S - \lambda_i v_i v_i^T$
- Return $XV$, where the columns of $V$ are $v_1, v_2, \ldots, v_M$. 
Clustering is an unsupervised machine learning technique, which given input data tries to divide them into some number of groups, or *clusters*.

The number of clusters might be given in advance, or should also be inferred.

When clustering documents, we usually use TF-IDF normalized so that each feature vector has length 1 (i.e., L2 normalization).
Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be a collection of $N$ input examples, each being a $D$-dimensional vector $\mathbf{x}_i \in \mathbb{R}^D$. Let $K$, the number of target clusters, be given.

Let $z_{i,k} \in \{0, 1\}$ be binary indicator variables describing whether an input example $\mathbf{x}_i$ is assigned to cluster $k$, and let each cluster be specified by a point $\mathbf{\mu}_1, \ldots, \mathbf{\mu}_K$, usually called the cluster center.

Our objective function $J$, which we aim to minimize, is

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{i,k} ||\mathbf{x}_i - \mathbf{\mu}_k||^2.$$
K-Means Clustering

**Input:** Input points $x_1, \ldots, x_N$, number of clusters $K$.

- Initialize $\mu_1, \ldots, \mu_K$ as $K$ random input points.
- Repeat until convergence (or until patience runs out):
  - Compute the best possible $z_{i,k}$. It is easy to see that the smallest $J$ is achieved by
    \[
    z_{i,k} = \begin{cases} 
    1 & \text{if } k = \arg\min_j \|x_i - \mu_j\|^2, \\
    0 & \text{otherwise}. 
    \end{cases}
    \]
  - Compute the best possible $\mu_k = \arg\min_{\mu} \sum_i z_{i,k} \|x_i - \mu\|^2$. By computing a derivative with respect to $\mu$, we get
    \[
    \mu_k = \frac{\sum_i z_{i,k} x_i}{\sum_i z_{i,k}}.
    \]
K-Means Clustering

Figure 9.1 of Pattern Recognition and Machine Learning.
K-Means Clustering

It is easy to see that:

- updating the cluster assignment $z_{i,k}$ decreases the loss $J$ or keeps it the same;
- updating the cluster centers again decreases the loss $J$ or keeps it the same.

K-Means clustering therefore converges to a local optimum.

However, it is quite sensitive to the starting initialization:

- It is common practice to run K-Means algorithm multiple times with different initialization and use the result with lowest $J$ (scikit-learn uses n_init=10 by default).
- Instead of using random initialization, k-means++ initialization scheme might be used, where the first cluster center is chosen randomly and others are chosen proportionally to the square of their distance to the nearest cluster center. It can be proven that with such initialization, the found solution has $O(\log K)$ approximation ratio in expectation.
K-Means Clustering

$K = 2$  \hspace{1cm} $K = 3$  \hspace{1cm} $K = 10$  \hspace{1cm} Original image

Figure 9.3 of Pattern Recognition and Machine Learning.
Gaussian Mixture vs K-Means

It could be useful to consider that different clusters might have different radii or even be ellipsoidal.

Different cluster analysis results on "mouse" data set:

Original Data  k-Means Clustering  EM Clustering

https://commons.wikimedia.org/wiki/File:ClusterAnalysis_Mouse.svg
Recall that

\[ \mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left( -\frac{(x - \mu)^2}{2\sigma^2} \right). \]

For \( D \)-dimensional vector \( x \), the multivariate Gaussian distribution takes the form

\[ \mathcal{N}(x|\mu, \Sigma) \overset{\text{def}}{=} \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right). \]

The biggest difference compared to the single-dimensional case is the covariance matrix \( \Sigma \), which is (in the non-degenerate case, which is the only one considered here) a symmetrical positive-definite matrix of size \( D \times D \).
If the covariance matrix is an identity, then the multivariate Gaussian distribution simplifies to

\[ \mathcal{N}(\mathbf{x} | \mu, I) = \frac{1}{\sqrt{(2\pi)^D}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)^T (\mathbf{x} - \mu) \right). \]

We can rewrite the exponent in this case to

\[ \mathcal{N}(\mathbf{x} | \mu, I) \propto \exp \left( -\frac{\|\mathbf{x} - \mu\|^2}{2} \right). \]

Therefore, the constant surfaces are concentric circles centered at the mean \( \mu \).

The same holds if the covariance is \( \sigma^2 I \), only the circles' diameter changes.
Multivariate Gaussian Distribution

Now consider a diagonal covariance matrix $\Lambda$. The exponent then simplifies to

$$
\mathcal{N}(\mathbf{x} | \mu, \Lambda) \propto \exp \left( - \sum_i \frac{1}{2 \Lambda_{ii}} (\mathbf{x}_i - \mu_i)^2 \right).
$$

The constant surfaces in this case are axis-aligned ellipses centered at the mean $\mu$ with size of the axes depending on the corresponding diagonal entries in the covariance matrix.
In the general case of a full covariance matrix, the fact that it is positive definite implies it has real positive eigenvalues $\lambda_i$. Considering the corresponding eigenvectors $u_i$, it can be shown that the constant surfaces are again ellipses centered at $\mu$, but this time rotated so that their axes are the eigenvectors $u_i$ with sizes $\lambda_i^{1/2}$.

Figure 2.7 of Pattern Recognition and Machine Learning.
Generally, we can rewrite a positive-definite matrix $\Sigma$ as $(U \Lambda^{1/2})(U \Lambda^{1/2})^T$, and then

$$x \sim \mathcal{N}(\mu, \Sigma) \iff x \sim \mu + U \Lambda^{1/2} \mathcal{N}(0, I).$$

Therefore, when sampling from a distribution with a full covariance matrix, we can sample from a standard multivariate $\mathcal{N}(0, I)$, scale by the eigenvalues of the covariance matrix, rotate according to the eigenvectors of the covariance matrix and finally shifting by $\mu$.

Note that different forms of covariance allows more generality, but also requires more parameters:

- the $\sigma^2 I$ has a single parameter,
- the $\Lambda$ has $D$ parameters,
- the full covariance matrix $\Sigma$ has $\binom{D+1}{2}$ parameters, i.e., $\Theta(D^2)$.
Let $x_1, x_2, \ldots, x_N$ be a collection of $N$ input examples, each being a $D$-dimensional vector $x_i \in \mathbb{R}^D$. Let $K$, the number of target clusters, be given.

Our goal is to represent the data as a Gaussian mixture, which is a combination of $K$ Gaussians in the form

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x | \mu_k, \Sigma_k).$$

Therefore, each cluster is parametrized as $\mathcal{N}(x | \mu_k, \Sigma_k)$. 
Gaussian Mixture

Figure 9.5 of Pattern Recognition and Machine Learning.

Figure 2.23 of Pattern Recognition and Machine Learning.