SVD, PCA, K-Means

Jindřich Libovický (reusing materials by Milan Straka)

📅 December 12, 2023
After this lecture you should be able to

- Theoretically explain Singular Value Decomposition (SVD), prove it exists and explain what the Eckart-Young theorem says
- Theoretically explain Principal Component Analysis (PCA) and say how it explains the variance in the data based on SVD
- Use SVD or PCA for dimensionality reduction, data visualization and data whitening
- Implement the $k$-means algorithm and use it for clustering
Unsupervised Machine Learning
Matrix Decompositions

\[ X = AB \]

- \( B \) tell us how to construct \( M \) using columns of \( A \)
- \( A \) tell us how to construct \( M \) using rows of \( B \)

\( X \) could be our (training) data matrix. If we managed to get decomposition with orthogonal columns/rows, it would tell us something like (statistical) independent parts the dataset consists of.

- Rows are data points
- Columns are features
Let $A \in \mathbb{C}^{N \times N}$ be an $N \times N$ matrix.

- A vector $v \in \mathbb{C}^N$ is a (right) eigenvector, if there exists an eigenvalue $\lambda \in \mathbb{C}$, such that

$$Av = \lambda v.$$

- If $A \in \mathbb{R}^{N \times N}$ is a real symmetric matrix, then it has $N$ real eigenvalues and $N$ real eigenvectors, which can be chosen to be orthonormal.

**Quick (almost) proof of orthogonality**

$Av_1 = \lambda_1 v_1$, we transpose both sides and get $(Av_1)^T = \lambda_1 v_1^T$

Multiply by $v_2$ from right, we get $v_1^T A^T v_2 = \lambda_1 v_1^T v_2$, but $A$ is symmetric and $A^T = A$, so

$v_1^T (Av_2) = \lambda_2 v_1^T v_2 \Rightarrow \lambda_2 v_1^T v_2 = \lambda_1 v_1^T v_2$, resulting in

$$(\lambda_2 - \lambda_1)(v_1^T v_2) = 0.$$
We can express real symmetric \( A \) using the **eigenvalue decomposition**

\[
A = V \Lambda V^T,
\]

where:

- \( V \) is a matrix, whose columns are the orthonormal eigenvectors \( v_1, v_2, \ldots, v_N \);
- \( \Lambda \) is a diagonal matrix with the eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_N \) on the diagonal.

**Quick derivation**

\[
AV = \Lambda V = V \Lambda
\]

\[
AVV^T = V \Lambda V^T
\]

Because \( V \) is orthonormal \( V^T = V^{-1} \), so \( VV^T = 1 \) and \( A = V \Lambda V^T \).
Singular Value Decomposition

Every (even a rectangular) matrix $X$ of dimension $m \times n$ and rank $r$ can be factorized as

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

- $U$ is a $m \times m$ orthonormal matrix
- $\Sigma$ is a $m \times n$ diagonal matrix with non-negative values, so-called singular values, chosen to be in decreasing order
- $V$ is a $n \times n$ orthonormal matrix

$$X V = U \Sigma \quad \Rightarrow \quad X v_k = \sigma_k u_k \quad \forall k = 1, \ldots, r$$

$$X = \begin{bmatrix} v_1 & \cdots & v_r & \cdots & v_n \end{bmatrix} = \begin{bmatrix} u_1 & \cdots & u_r & \cdots & u_m \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & 0 \end{bmatrix}$$
Assuming SVD exists, we can write $U$ and $V$ as eigenvector decomposition of row and column similarities:

$$X X^T = (U \Sigma V^T)(U \Sigma V^T)^T = U \Sigma (V^T V) \Sigma^T U^T = U \Sigma \Sigma^T U^T$$

$$X^T X = (U \Sigma V^T)^T (U \Sigma V^T) = V \Sigma (U^T U) \Sigma^T V^T = V \Sigma^T \Sigma V^T$$

Let's take $V = [v_1, \ldots, v_r]$ orthonormal eigenvectors of $X^T X$ and set $\sigma_k = \sqrt{\lambda_k}$, then, $X^T X v_k = \sigma_k^2 v_k$. ($\lambda_k \geq 0$ because $X X^T$ is positive semi-definite.)

The decomposition says that $X v_k = \sigma_k u_k \Rightarrow u_k = \frac{X v_k}{\sigma_k}$. To make it work, we need to show that $u_k$ is indeed an eigenvector of $X X^T$ with the same eigenvalue $\lambda_k$.

$$X X^T u_k = X X^T \left( \frac{X v_k}{\sigma_k} \right) = X \left( \frac{X^T X v_k}{\sigma_k} \right) = X \frac{\sigma_k^2 v_k}{\sigma_k} = \sigma_k^2 \left( \frac{X v_k}{\sigma_k} \right) = \sigma_k^2 u_k$$

def. of $u_k$

$u_k$ is eigenvector of $X^T X$
Interpretation of SVD

- Vectors of $U$ are the components rows consists of, vectors of $V$ are the same for the columns.
- It defines a decomposition of $X$ (with rank $r$) as a sum of rank 1 matrices of dimension $m \times n$

$$X = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_r u_r v_r^T$$

- Reduced version of SVD: We can throw away $\sigma_k$ for $k > r$ and use smaller $U$ and $V$
- $\sigma$ are in the decreasing order $\Rightarrow$ we can approximate $X$ by taking $k < \min(m, n)$

$$\tilde{X} = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

Eckart-Young theorem: This is the best rank $k$ approximation w.r.t. Frobenius norm (we flatten the matrix to a vector and do $L^2$ norm).
Eckart-Young Theorem

$X \in \mathbb{R}^{n \times m}$ and $X_k = \sigma_1 u_1 v_1^T + \ldots + \sigma_k u_k v_k^T$ its approximation using SVD. For each $B \in \mathbb{R}^{n \times m}$ of rank $k$

$$||X - X_k||_F \leq ||X - B||_F.$$

**Argument why it is a good idea**

- $||X||_F = \sqrt{\sum_i^n \sum_j^m x_{ij}^2} = \sqrt{\text{trace}(X^T X)}$ (trace is the sum of the diagonal)

- Multiplying by orthonormal matrix does not change the norm

$$||UA||_F^2 = \text{trace}((UA)^T UA) = \text{trace}(A^T \underbrace{U^T UA}_I) = \text{trace}(A^T A) = ||A||_F^2$$

- The Frobenius norm is the $L^2$ norm of the diagonal of $\Sigma$

- The best strategy to keep the most of the norm is removing the smallest values
## Image Compression using SVD

<table>
<thead>
<tr>
<th>Components</th>
<th>Image Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>400 components</td>
<td><img src="https://via.placeholder.com/150" alt="Image 1" /></td>
</tr>
<tr>
<td>200 components</td>
<td><img src="https://via.placeholder.com/150" alt="Image 2" /></td>
</tr>
<tr>
<td>100 components</td>
<td><img src="https://via.placeholder.com/150" alt="Image 3" /></td>
</tr>
<tr>
<td>50 components</td>
<td><img src="https://via.placeholder.com/150" alt="Image 4" /></td>
</tr>
<tr>
<td>10 components</td>
<td><img src="https://via.placeholder.com/150" alt="Image 5" /></td>
</tr>
</tbody>
</table>

The images show the effect of reducing the number of components using Singular Value Decomposition (SVD) for image compression. As the number of components decreases, the images become coarser, and the compression factor increases.
SVD in Recommender Systems

- You have a matrix of what users like what content on a streaming platform.
- A user can be represented by a vector of content they liked, content can be represented by a vector of user that liked it.
- Such a matrix is huge and noisy: SVD can be used to reduce the noise (throw away small singular values).
- Low-dimensional representation of users and content in terms “eigenusers” and “eigencontent”.
- In practice, slightly modified version of SVD are used.
So far, SVD had geometric interpretation, let's add statistical interpretation.

When we apply SVD on mean-centered data $\mathbf{X} - \bar{\mathbf{x}}$, singular values get a new interpretation: components explaining variability in the data.

$$\| \mathbf{X} - \bar{\mathbf{x}} \|_F^2 = \text{trace} \left( (\mathbf{X} - \bar{\mathbf{x}})^T (\mathbf{X} - \bar{\mathbf{x}}) \right) = N \sum_i^D \text{Var}(\mathbf{X}(:,i))$$

Approximating the matrix in terms of Frobenius norm means keeping the most variance from the data. Components are ordered by how much variability in the data they capture.

Let $\mathbf{S} = \frac{1}{N} (\mathbf{X} - \bar{\mathbf{x}})^T (\mathbf{X} - \bar{\mathbf{x}})$, then PCA of $\mathbf{X}$ are the eigenvectors of $\mathbf{S}$, i.e., the $\mathbf{V}$ matrix of the SVD decomposition of $\mathbf{X} - \bar{\mathbf{x}}$.

Note the $\frac{1}{N}$ term scales down the eigenvalues compared to SVD, but keeps the eigenvectors unchanged.
Principle components in data sampled from a 2D Gaussian. $U \Sigma V$ is SVD decomposition of mean-centered data matrix $X$.

- Vectors of $(\Sigma V)$ define the directions of the components, so called **loadings**.
- Each vector of $U$ represents one corresponding centered example $X \in X - \bar{x}$ as distances in a coordinate system given by the loadings.

https://en.wikipedia.org/wiki/Principal_component_analysis#/media/File:GaussianScatterPCA.svg
Principal Component Analysis

The principal component analysis, PCA, is a linear transformation 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.

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.
The PCA formula allows us to perform **whitening** aka **sphering**, which is a linear transformation of the given data so that the resulting dataset has zero mean and an identity covariance matrix.

Notably, if \( \mathbf{V} \) are the eigenvectors of \( \mathbf{S} \) and \( \Sigma^2 \) is the diagonal matrix of the corresponding eigenvalues (i.e., \( \mathbf{SV} = \mathbf{V} \Sigma^2 \)), we can define the transformed data as

\[
\mathbf{z}_i \overset{\text{def}}{=} \Sigma^{-1} \mathbf{V}^T (\mathbf{x}_i - \bar{\mathbf{x}}).
\]

Then, the mean of \( \mathbf{z}_i \) is zero and the covariance is given by

\[
\frac{1}{N} \sum_i \mathbf{z}_i \mathbf{z}_i^T = \frac{1}{N} \sum_i \Sigma^{-1} \mathbf{V}^T (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{V} \Sigma^{-1}
\]

\[
= \Sigma^{-1} \mathbf{V}^T \mathbf{S} \mathbf{V} \Sigma^{-1} = \Sigma^{-1} \Sigma^2 \Sigma^{-1} = \mathbf{I}.
\]
The red components that are askew and different length become the $x$ and $y$ axis and have the same length.
Data Visualization

Word embeddings from neural machine translation.

Note that PCA does not have access to supervised labels, so it may not give a solution favorable for further classification. PCA = projecting on the magenta line, which does not help the classification.
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 an absolute value strictly larger than absolute values of all other eigenvalues) and the corresponding eigenvector (it is used for example to compute PageRank). It works as follows:

**Input**: Real symmetric 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 $\lambda$. 
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 \(a_1 u_1 + a_2 u_2 + a_3 u_1 \ldots\), then \(A v\) is in the basis of the eigenvectors:

\[
Av = \lambda_1 a_1 u_1 + \lambda_2 a_2 u_2 + \lambda_3 a_3 u_3 \ldots
\]

\[
A^k v = \lambda_1^k a_1 u_2 + \lambda_2^k a_2 u_2 + \lambda_3^k a_3 u_3 \ldots = \lambda_1^k \left(a_1 u_1 + \frac{\lambda_2^k}{\lambda_1^k} a_2 u_2 + \frac{\lambda_3^k}{\lambda_1^k} a_3 u_3 + \ldots\right)
\]

Coordinates \(\frac{\lambda_i^k}{\lambda_1^k}\) go to zero with \(k \to \infty\). Normalization during the algorithm prevents the \(\lambda_1^k\) term from exploding.

If the initial \(v\) had a nonzero first coordinate \(a_1\) (which has probability very close to 1), then repeated multiplication with \(A\) converges 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 $\lambda_1$”. Consider $A - \lambda_1 v_1 v_1^T$:

- 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 as multiplying $A$:
  \[
  (A - \lambda_1 v_1 v_1^T)v_i = Av_i - \lambda_1 v_1 v_1^T v_i = Av_i.
  \]

Therefore, $A - \lambda_1 v_1 v_1^T$ has the same set of eigenvectors and eigenvalues, except for $v_1$, which now has eigenvalue 0.
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):
    - $v_i \leftarrow Sv_i$
    - $\lambda_i \leftarrow \|v_i\|$
    - $v_i \leftarrow v_i / \lambda_i$
    - $S \leftarrow S - \lambda_i v_i v_i^T$

- 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 we should infer it.

When clustering documents, we usually normalize TF-IDF so that each feature vector has length 1 (i.e., L2 normalization), because then

\[
1 - \text{cosine similarity}(x, y) = \frac{1}{2} \|x - y\|^2.
\]
K-Means Clustering

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 $\mu_1, \ldots, \mu_K$, usually called the cluster center.

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

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{i,k} \| \mathbf{x}_i - \mu_k \|^2.$$
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 the 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 this initialization, the solution has $O(\log K)$ approximation ratio in expectation.
K-Means Clustering

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

![ClusterAnalysis_Mouse.svg](https://commons.wikimedia.org/wiki/File:ClusterAnalysis_Mouse.svg)
After this lecture you should be able to

- Theoretically explain Singular Value Decomposition (SVD), prove it exists and explain what the Eckart-Young theorem says
- Theoretically explain Principal Component Analysis (PCA) and say how it explains the variance in the data based on SVD
- Use SVD or PCA for dimensionality reduction, data visualization and data whitening
- Implement the $k$-means algorithm and use it for clustering