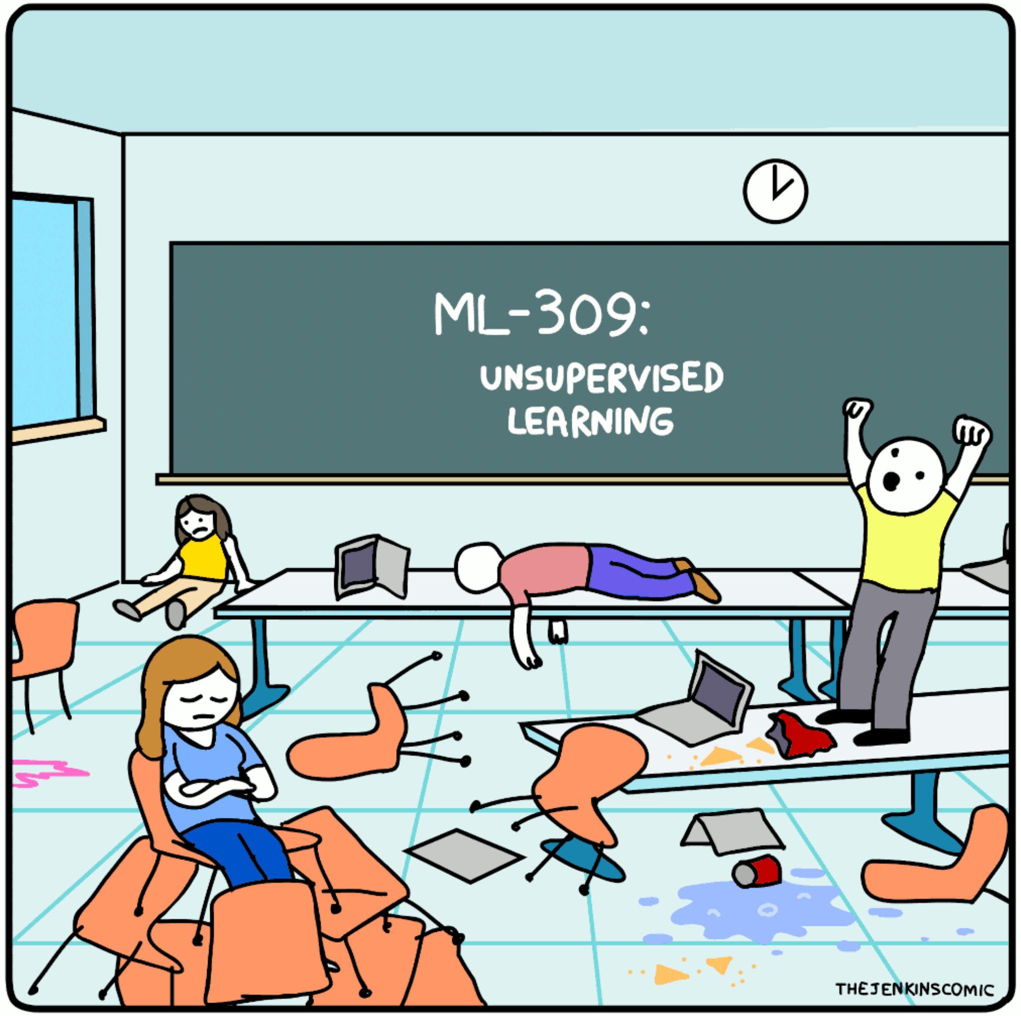
PCA, K-Means

Milan Straka

December 12, 2022

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Unsupervised Machine Learning



https://thejenkinscomic.files.wordpress.com/2022/09/screen-shot-2022-09-22-at-9.49.35-pm.png

Linear Algebra Refresh – Eigenvalues and Eigenvectors

Let $\boldsymbol{A} \in \mathbb{C}^{N\times N}$ be an $N \times N$ matrix.

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

$$\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}.$$

- If $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ is a real symmetric matrix, then it has N real eigenvalues and N real eigenvectors, which can be chosen to be *orthonormal*, and we can express $\boldsymbol{A}$ using the **eigenvalue decomposition**

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T,$$

  where:
  - $\boldsymbol{V}$ is a matrix, whose columns are the eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N$;
  - $\boldsymbol{\Lambda}$ is a diagonal matrix with the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_N$ on the diagonal.

Linear Algebra Refresh – Positive Definiteness

Let $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ be a real symmetric matrix. Then if for all $\boldsymbol{x} \neq \mathbf{0}$:

- $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0$, the matrix is called **positive definite**.

  Note that this condition is equivalent to all eigenvalues being **positive**.
- $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geq 0$, the matrix is called **positive semi-definite**.

  This condition is equivalent to all eigenvalues being **nonnegative**.
- $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} < 0$, the matrix is called **negative definite**.

  This condition is equivalent to all eigenvalues being **negative**.
- $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \leq 0$, the matrix is called **negative semi-definite**.

  This condition is equivalent to all eigenvalues being **nonpositive**.

Note that we can compute a "square root" of a positive (semi-)definite matrix, because if $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$, then for $\boldsymbol{\Lambda}^{1/2}\boldsymbol{V}^T$ we get

$$(\boldsymbol{\Lambda}^{1/2}\boldsymbol{V}^T)^T\boldsymbol{\Lambda}^{1/2}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Lambda}^{1/2}\boldsymbol{\Lambda}^{1/2}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T = \boldsymbol{A}.$$

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.

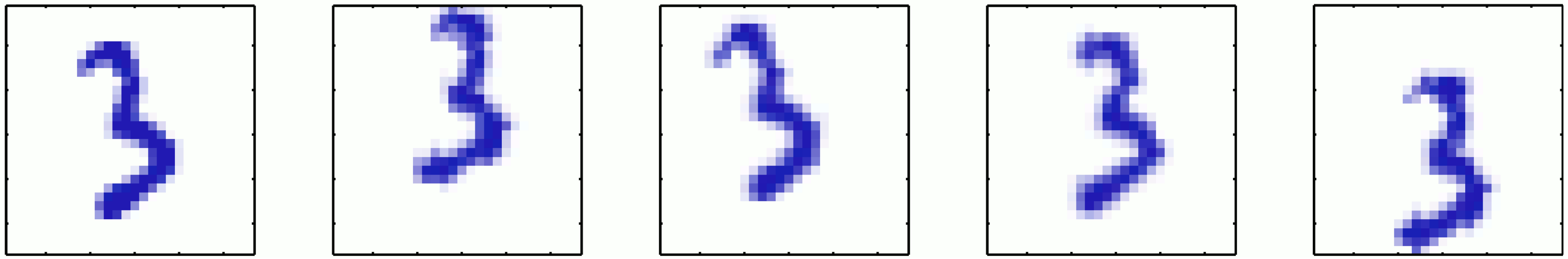


Figure 12.1 of Pattern Recognition and Machine Learning.

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*.

Principal Component Analysis

We start by defining the PCA in two ways.

Maximum Variance Formulation

Given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ with $\boldsymbol{x}_i \in \mathbb{R}^D$, 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 $\boldsymbol{u}_1$, and because only the direction of $\boldsymbol{u}_1$ matters, we assume that $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$.

The projection of $\boldsymbol{x}_i$ to $\boldsymbol{u}_1$ is given by $(\boldsymbol{u}_1^T \boldsymbol{x}_i)\boldsymbol{u}_1$, because the vectors $\boldsymbol{u}_1$ and $\boldsymbol{x}_i - (\boldsymbol{u}_1^T \boldsymbol{x}_i)\boldsymbol{u}_1$ are orthogonal:

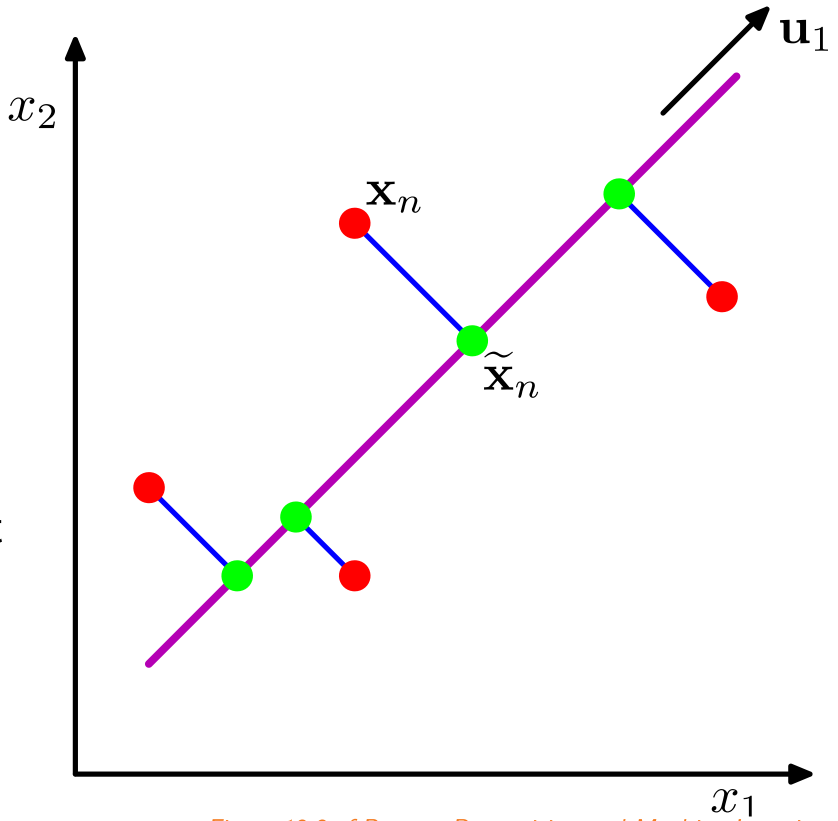


Figure 12.2 of Pattern Recognition and Machine Learning.

$$\boldsymbol{u}_1^T \big(\boldsymbol{x}_i - (\boldsymbol{u}_1^T \boldsymbol{x}_i)\boldsymbol{u}_1\big) = \boldsymbol{u}_1^T \boldsymbol{x}_i - (\boldsymbol{u}_1^T \boldsymbol{x}_i)\boldsymbol{u}_1^T \boldsymbol{u}_1 = 0.$$

Principal Component Analysis

We therefore use $\boldsymbol{u}_1^T \boldsymbol{x}_i$ as the projection of $\boldsymbol{x}_i$. If we define $\bar{\boldsymbol{x}} = \sum_i \boldsymbol{x}_i / N$, the mean of the projected data is $\boldsymbol{u}_1^T \bar{\boldsymbol{x}}$, and the variance is given by

$$\frac{1}{N} \sum_{i=1}^{N} \big(\boldsymbol{u}_1^T \boldsymbol{x}_i - \boldsymbol{u}_1^T \bar{\boldsymbol{x}}\big)^2 = \boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1,$$

where $\boldsymbol{S}$ is the data covariance matrix defined as

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

We can write the data covariance matrix in matrix form as $\boldsymbol{S} = \frac{1}{N}\big(\boldsymbol{X} - \bar{\boldsymbol{x}}\big)^T \big(\boldsymbol{X} - \bar{\boldsymbol{x}}\big)$.

If the original data is centered (it has zero mean), then $\boldsymbol{S} = \frac{1}{N} \boldsymbol{X}^T \boldsymbol{X}$, which we have already encountered.

Principal Component Analysis

To maximize $\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1$, we need to include the constraint $\boldsymbol{u}_1^T \boldsymbol{u}_1$ by introducing a Lagrange multiplier λ_1 for the constraint $\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1 = 0$ and then maximizing the Lagrangian

$$\mathcal{L} = \boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 - \lambda_1 \big(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1\big).$$

By computing a derivative with respect to $\boldsymbol{u}_1$, we get

$$\boldsymbol{S} \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1.$$

Therefore, $\boldsymbol{u}_1$ must be an eigenvector of $\boldsymbol{S}$ corresponding to eigenvalue λ_1.

Because the value to maximize $\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1$ is then $\boldsymbol{u}_1^T \lambda_1 \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1^T \boldsymbol{u}_1 = \lambda_1$, the maximum is attained for eigenvector $\boldsymbol{u}_1$ corresponding to the largest eigenvalue λ_1.

The eigenvector $\boldsymbol{u}_1$ 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.

Minimum Error Formulation

Assume $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D$ is some orthonormal set of vectors, therefore, $\boldsymbol{u}_i^T \boldsymbol{u}_j = \big[i = j\big] = \delta_{i,j}$.

Every $\boldsymbol{x}_i$ can be then expressed using this basis as

$$\boldsymbol{x}_i = \sum_j (\boldsymbol{x}_i^T \boldsymbol{u}_j) \boldsymbol{u}_j,$$

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 approximate the data by the first M basis vectors:

$$\tilde{\boldsymbol{x}}_i = \sum_{j=1}^{M} z_{i,j} \boldsymbol{u}_j + \sum_{j=M+1}^{D} b_j \boldsymbol{u}_j.$$

Principal Component Analysis

We now choose the vectors $\boldsymbol{u}_j$, coordinates $z_{i,j}$ and biases b_j to minimize the approximation error, which we measure as a loss

$$L = \frac{1}{N} \sum\nolimits_{i=1}^{N} \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2.$$

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

$$z_{i,j} = \boldsymbol{x}_i^T \boldsymbol{u}_j, \quad b_j = \bar{\boldsymbol{x}}^T \boldsymbol{u}_j.$$

Therefore, we can rewrite the loss as

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

Analogously, we can minimize L by choosing the eigenvectors of $D - M$ smallest eigenvalues.

PCA Applications – Data Compression

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

$$\tilde{\boldsymbol{x}}_i = \sum_{j=1}^{M} (\boldsymbol{x}_i^T \boldsymbol{u}_j)\boldsymbol{u}_j + \sum_{j=M+1}^{D} (\bar{\boldsymbol{x}}^T \boldsymbol{u}_j)\boldsymbol{u}_j = \bar{\boldsymbol{x}} + \sum_{j=1}^{M} (\boldsymbol{x}_i^T \boldsymbol{u}_j - \bar{\boldsymbol{x}}^T \boldsymbol{u}_j)\boldsymbol{u}_j.$$

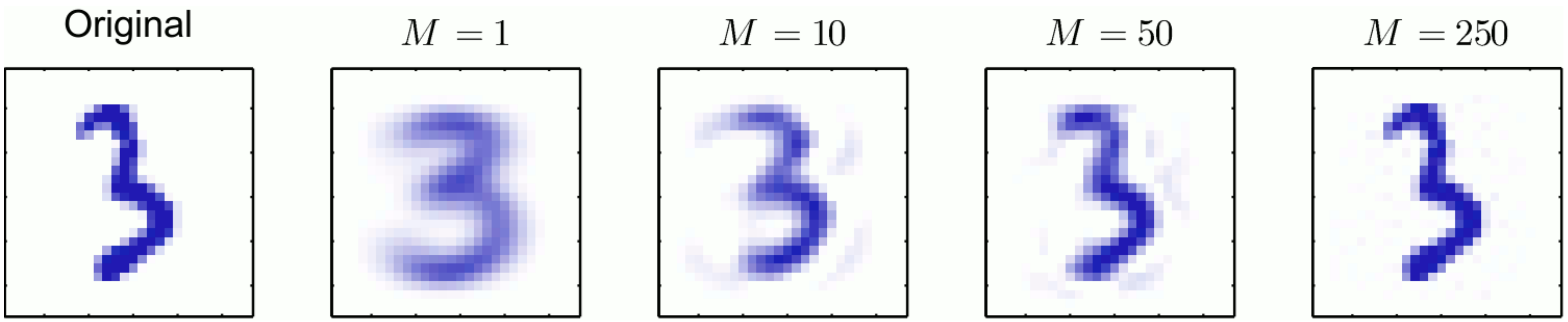
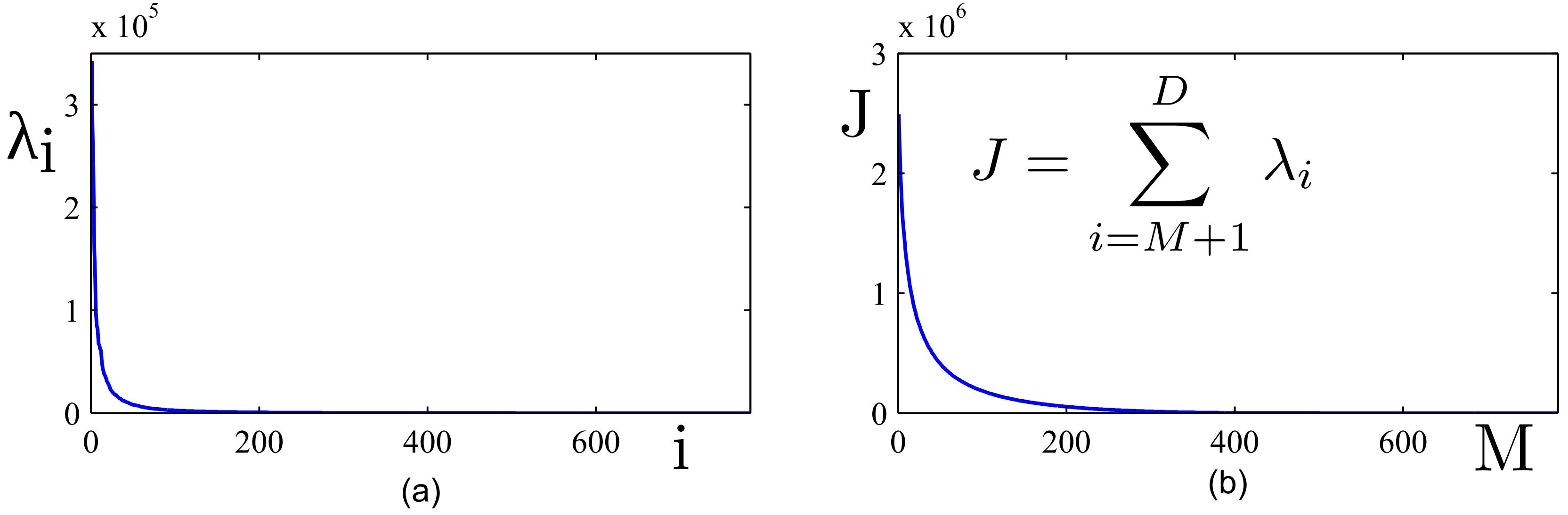


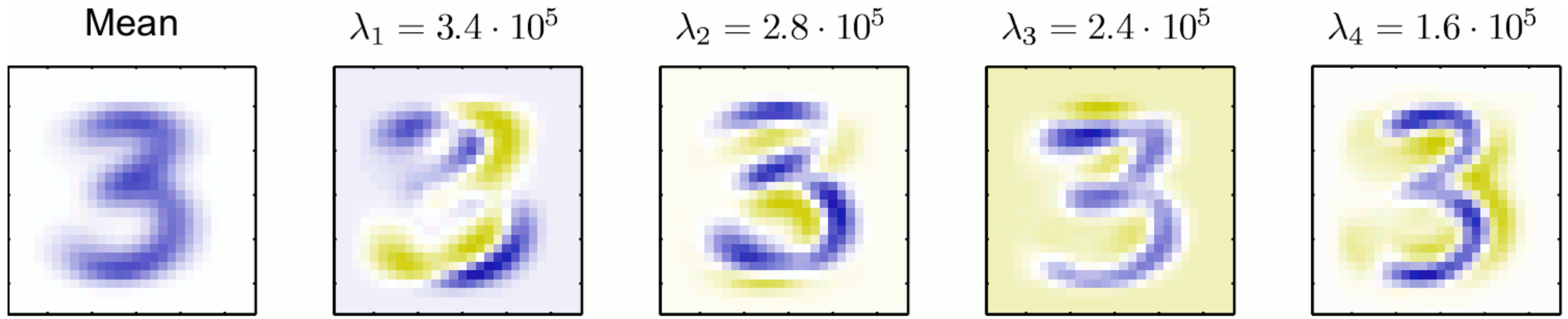
Figure 12.5 of Pattern Recognition and Machine Learning.

PCA Applications – Data Compression



$$J = \sum_{i=M+1}^{D} \lambda_i$$

Figure 12.4 of Pattern Recognition and Machine Learning.



Mean
$\lambda_1 = 3.4 \cdot 10^5$
$\lambda_2 = 2.8 \cdot 10^5$
$\lambda_3 = 2.4 \cdot 10^5$
$\lambda_4 = 1.6 \cdot 10^5$

Figure 12.3 of Pattern Recognition and Machine Learning.

PCA Applications – Whitening aka Sphering

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 $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{S}$ and $\boldsymbol{\Lambda}$ is the diagonal matrix of the corresponding eigenvalues (i.e., $\boldsymbol{S}\boldsymbol{U} = \boldsymbol{U}\boldsymbol{\Lambda}$), we can define the transformed data as

$$\boldsymbol{z}_i \stackrel{\text{def}}{=} \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T(\boldsymbol{x}_i - \bar{\boldsymbol{x}}).$$

Then, the mean of $\boldsymbol{z}_i$ is zero and the covariance is given by

$$\begin{aligned}\frac{1}{N}\sum_i \boldsymbol{z}_i\boldsymbol{z}_i^T &= \frac{1}{N}\sum_i \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T\boldsymbol{U}\boldsymbol{\Lambda}^{-1/2} \\ &= \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\boldsymbol{S}\boldsymbol{U}\boldsymbol{\Lambda}^{-1/2} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{\Lambda}\boldsymbol{\Lambda}^{-1/2} = \boldsymbol{I}.\end{aligned}$$

PCA Applications – Whitening or Sphering

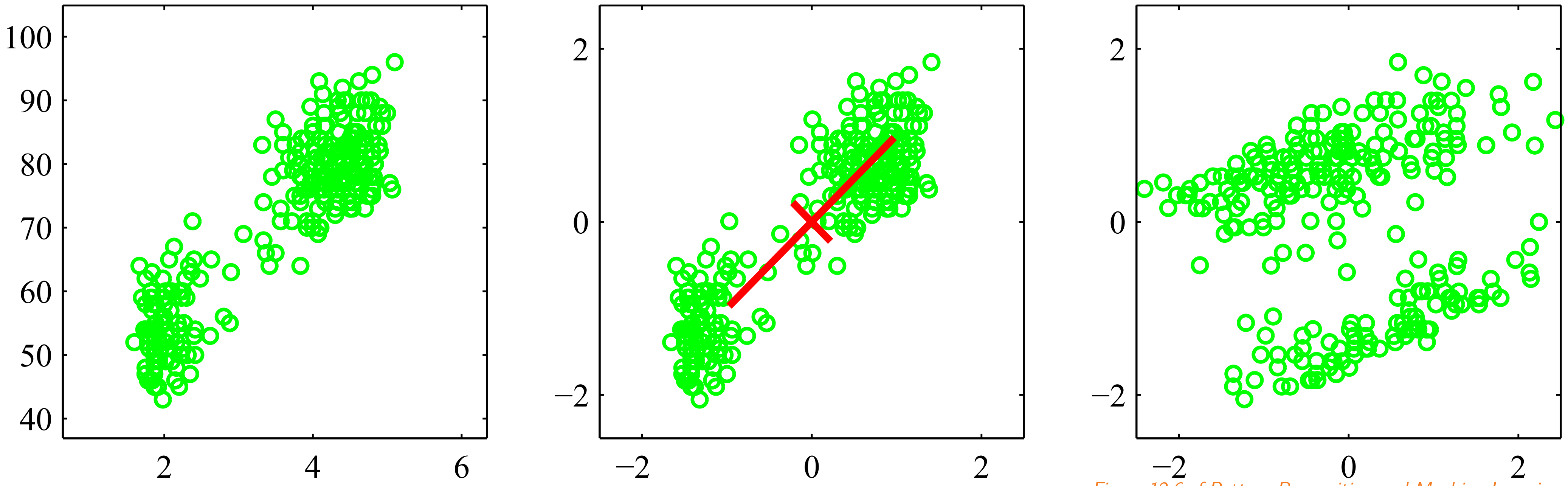


Figure 12.6 of Pattern Recognition and Machine Learning.

PCA versus Supervised ML

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

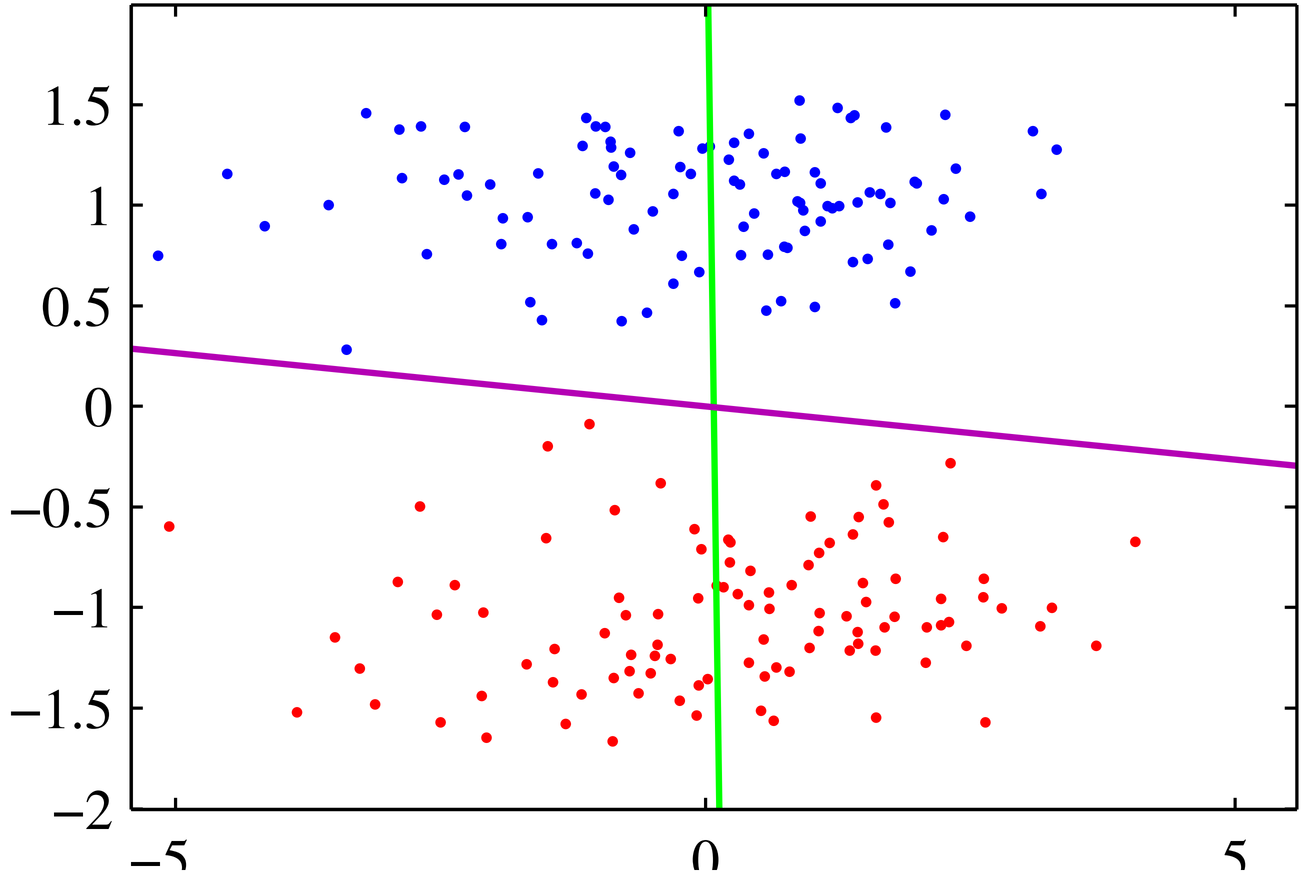


Figure 12.7 of Pattern Recognition and Machine Learning.

Principal Component Analysis and MLPs

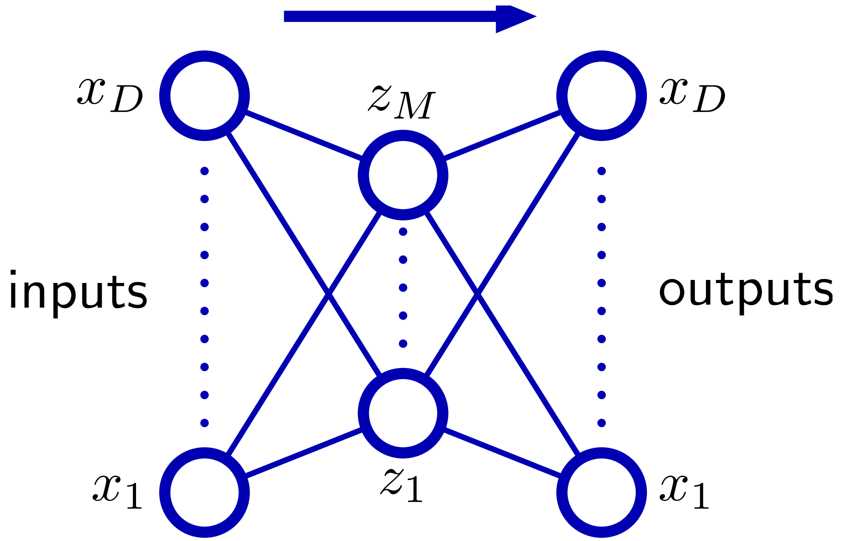


Figure 12.8 of Pattern Recognition and Machine Learning.

It can be proven that if we construct an MLP *autoencoder*, which is a model trying to reconstruct input as close as possible, then even if the hidden layer uses nonlinear activation, the solution to an MSE loss is a projection onto the M-dimensional subspace defined by the first M principal components (but is not necessary orthonormal or orthogonal).

Principal Component Analysis and MLPs

However, nonlinear PCA can be achieved, if both the *encoder* and the *decoder* are nonlinear.

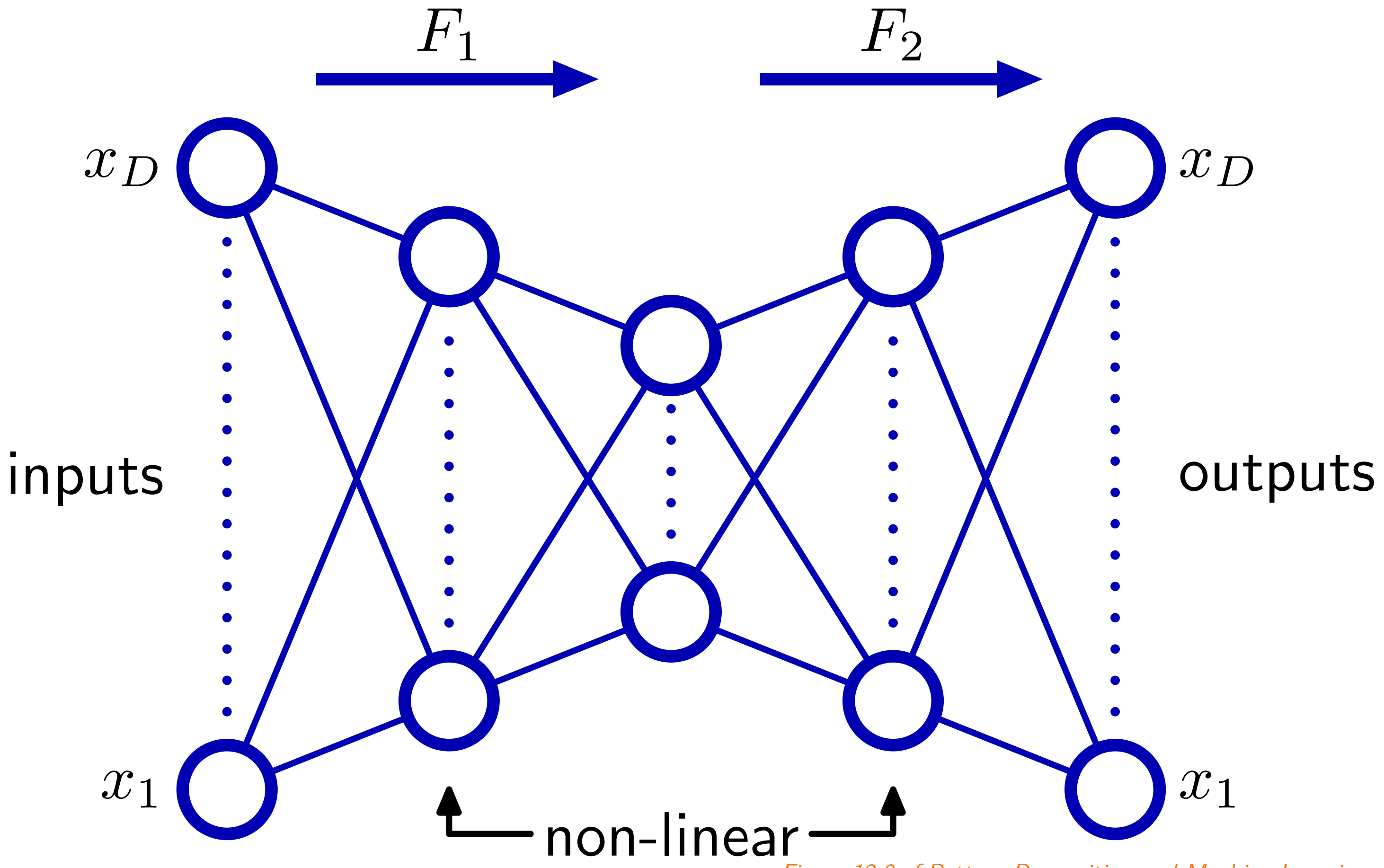


Figure 12.9 of Pattern Recognition and Machine Learning.

Computing PCA

There are two frequently used algorithms for performing PCA.

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

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

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

Therefore,

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

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

Computing PCA — The Power Iteration Algorithm

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 $\boldsymbol{A}$ with a dominant eigenvalue.

Output: The dominant eigenvalue λ and the corresponding eigenvector $\boldsymbol{v}$, 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):
  - $\boldsymbol{v} \leftarrow \boldsymbol{A}\boldsymbol{v}$
  - $\lambda \leftarrow \|\boldsymbol{v}\|$
  - $\boldsymbol{v} \leftarrow \boldsymbol{v}/\lambda$

If the algorithm converges, then $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{v}/\lambda$, so $\boldsymbol{v}$ is an eigenvector with eigenvalue λ.

Computing PCA — The Power Iteration Algorithm

In order to analyze the convergence, let $(\lambda_1, \lambda_2, \lambda_3, \ldots)$ be the eigenvalues of $\boldsymbol{A}$, 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 $\boldsymbol{v}$ in the basis of the eigenvectors as $(a_1, a_2, a_3, \ldots)$, then $\boldsymbol{Av}/\lambda_1$ is in the basis of the eigenvectors:

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

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

If the initial $\boldsymbol{v}$ had a nonzero first coordinate a_1 (which has probability very close to 1), then repeated multiplication with $\boldsymbol{A}$ converges to the eigenvector corresponding to λ_1.

Computing PCA — The Power Iteration Algorithm

After we get the largest eigenvalue λ_1 and its eigenvector $\boldsymbol{v}_1$, we can modify the matrix $\boldsymbol{A}$ to "remove the eigenvalue λ_1". Consider $\boldsymbol{A} - \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T$:

- multiplying it by $\boldsymbol{v}_1$ returns zero:

$$\big(\boldsymbol{A} - \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T\big)\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1 - \lambda_1 \boldsymbol{v}_1 \underbrace{\boldsymbol{v}_1^T \boldsymbol{v}_1}_{1} = 0,$$

- multiplying it by other eigenvectors $\boldsymbol{v}_i$ gives the same result as multiplying $\boldsymbol{A}$:

$$\big(\boldsymbol{A} - \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T\big)\boldsymbol{v}_i = \boldsymbol{A}\boldsymbol{v}_i - \lambda_1 \boldsymbol{v}_1 \underbrace{\boldsymbol{v}_1^T \boldsymbol{v}_i}_{0} = \boldsymbol{A}\boldsymbol{v}_i.$$

Therefore, $\boldsymbol{A} - \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T$ has the same set of eigenvectors and eigenvalues, except for $\boldsymbol{v}_1$, which now has eigenvalue 0.

Computing PCA — The Power Iteration Algorithm

We are now ready to formulate the complete algorithm for computing the PCA.

Input: Matrix $\boldsymbol{X}$, desired number of dimensions M.

- Compute the mean $\boldsymbol{\mu}$ of the examples (the rows of $\boldsymbol{X}$).
- Compute the covariance matrix $\boldsymbol{S} \leftarrow \frac{1}{N}\big(\boldsymbol{X} - \boldsymbol{\mu}\big)^T\big(\boldsymbol{X} - \boldsymbol{\mu}\big)$.
- for i in $\{1, 2, \ldots, M\}$:
    - Initialize $\boldsymbol{v}_i$ randomly.
    - Repeat until convergence (or for a fixed number of iterations):
        - $\boldsymbol{v}_i \leftarrow \boldsymbol{S}\boldsymbol{v}_i$
        - $\lambda_i \leftarrow \|\boldsymbol{v}_i\|$
        - $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i / \lambda_i$
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} - \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T$
- Return $\boldsymbol{X}\boldsymbol{V}$, where the columns of $\boldsymbol{V}$ are $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_M$.

Clustering

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

$$1 - \operatorname{cosine\,similarity}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

K-Means Clustering

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ be a collection of N input examples, each being a D-dimensional vector $\boldsymbol{x}_i \in \mathbb{R}^D$. 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 $\boldsymbol{x}_i$ is assigned to cluster k, and let each cluster be specified by a point $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, usually called the cluster **center**.

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

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

K-Means Clustering

Input: Input points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, number of clusters K.

- Initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ 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

$$z_{i,k} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2, \\ 0 & \text{otherwise.} \end{cases}$$

  - Compute the best possible $\boldsymbol{\mu}_k = \arg\min_{\boldsymbol{\mu}} \sum_i z_{i,k} \|\boldsymbol{x}_i - \boldsymbol{\mu}\|^2$. By computing a derivative with respect to $\boldsymbol{\mu}$, we get

$$\boldsymbol{\mu}_k = \frac{\sum_i z_{i,k} \boldsymbol{x}_i}{\sum_i z_{i,k}}.$$

K-Means Clustering

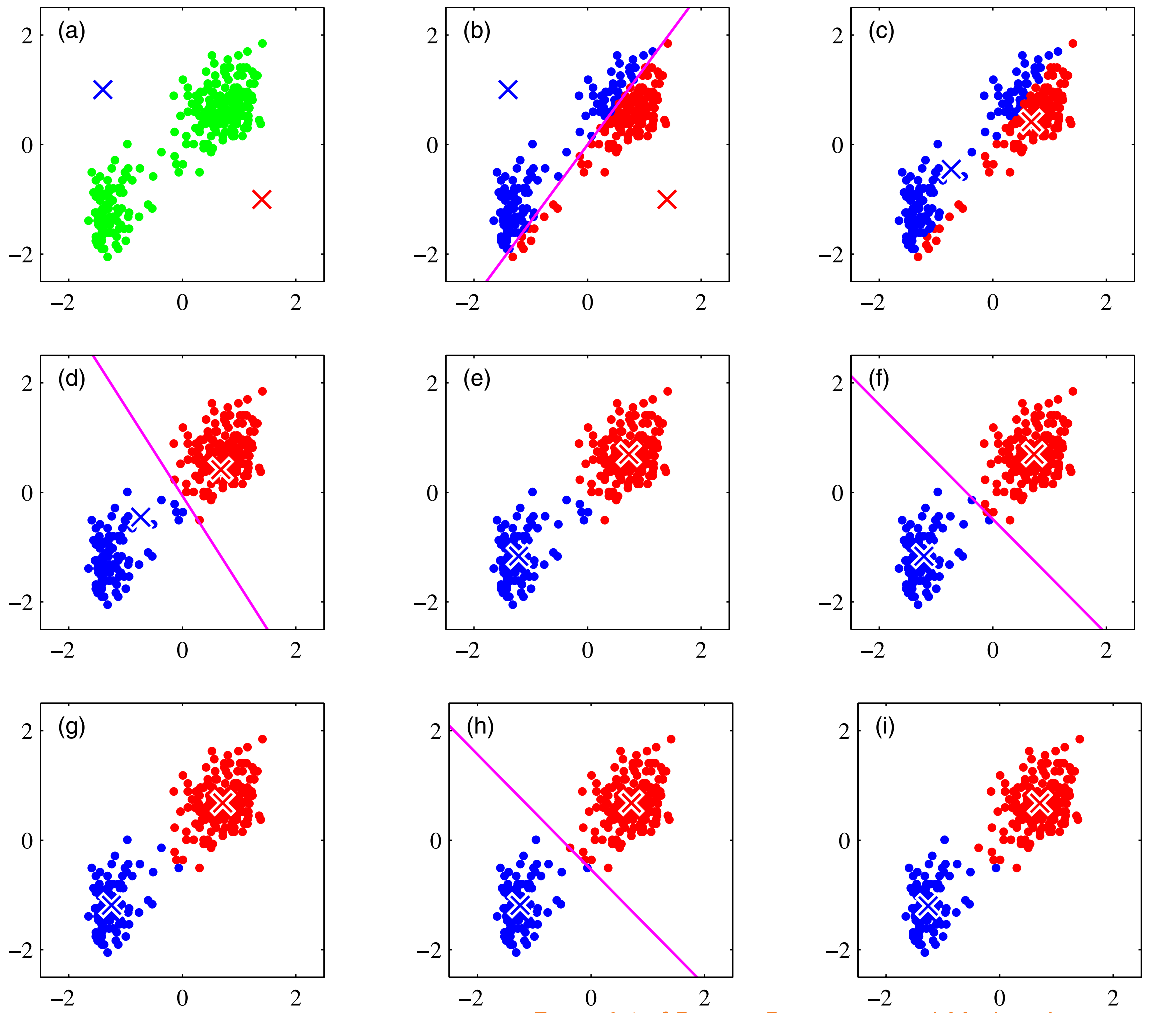


Figure 9.1 of Pattern Recognition and Machine Learning.

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.

Plot of the cost function J given by (9.1) after each E step (blue points) and M step (red points) of the K-means algorithm for the example shown in Figure 9.1. The algorithm has converged after the third M step, and the final EM cycle produces no changes in either the assignments or the prototype vectors.

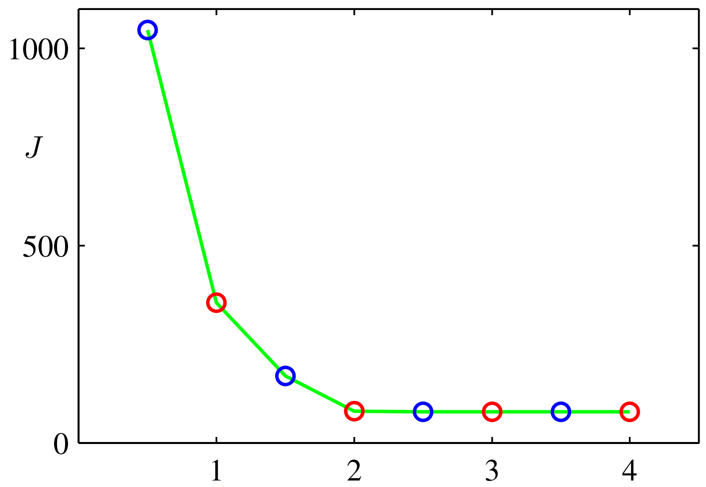


Figure 9.2 of Pattern Recognition and Machine Learning.

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 $\mathcal{O}(\log K)$ approximation ratio in expectation.

K-Means Clustering

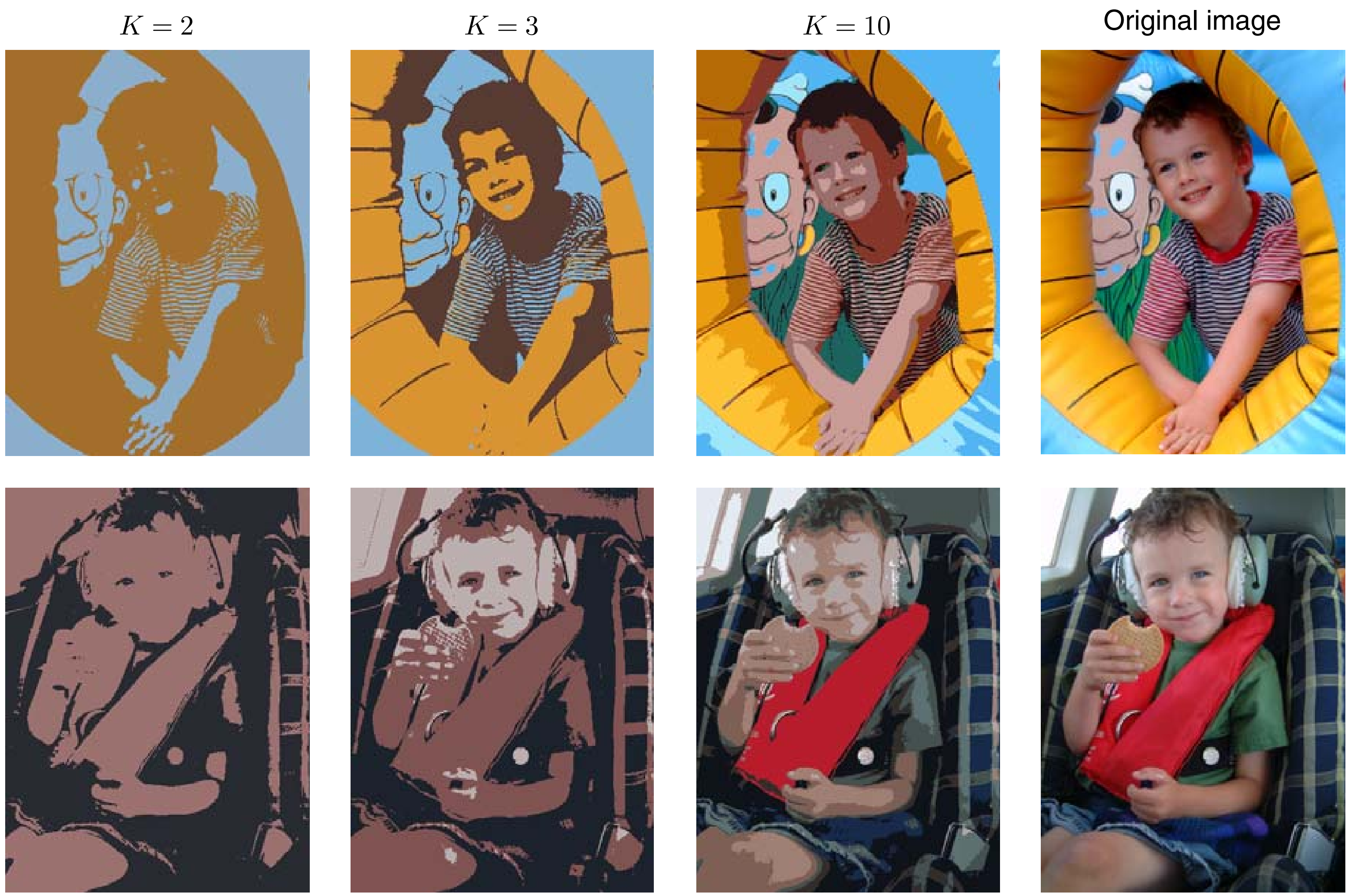
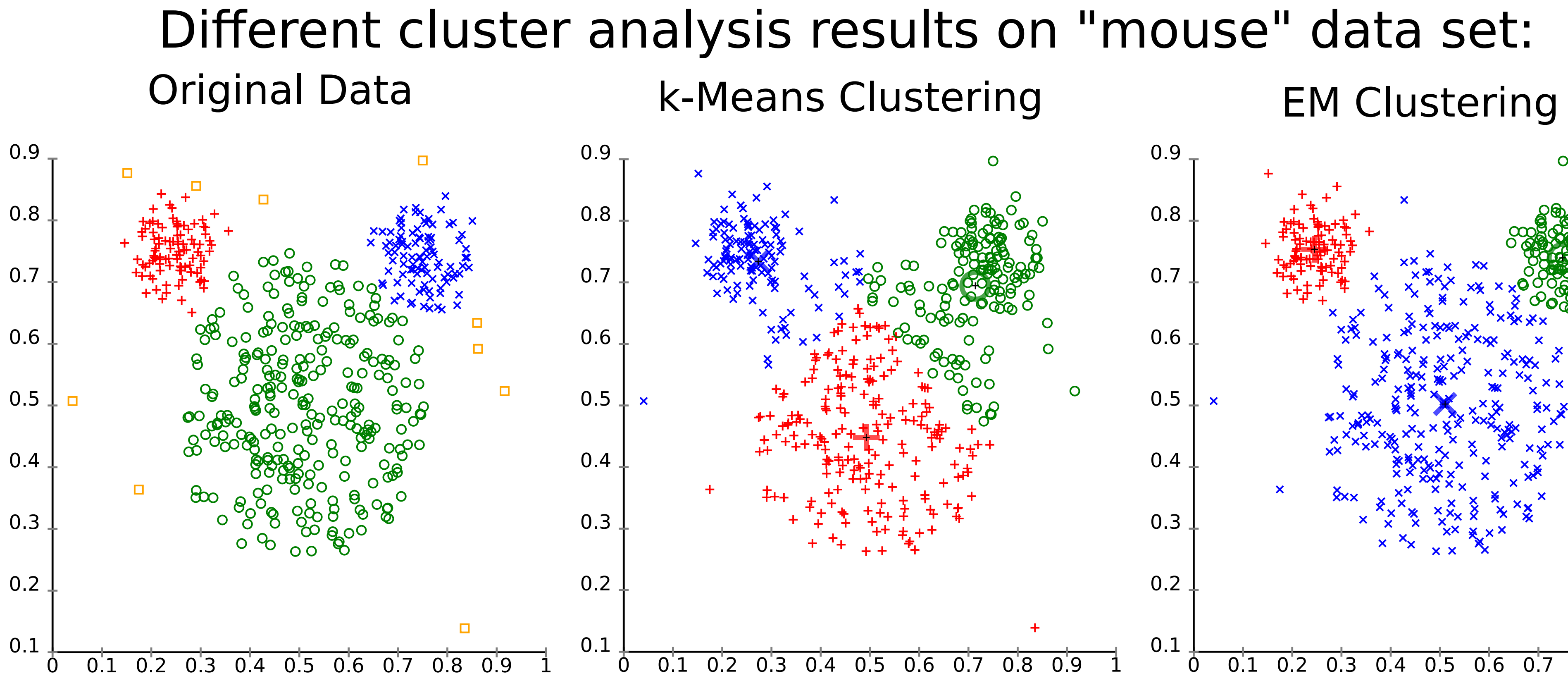


Figure 9.3 of Pattern Recognition and Machine Learning.

Gaussian Mixture vs K-Means

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



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