PCA, K-Means, Gaussian Mixture

Milan Straka

December 14, 2020

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

Unsupervised Machine Learning

Principal Component Analysis

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.

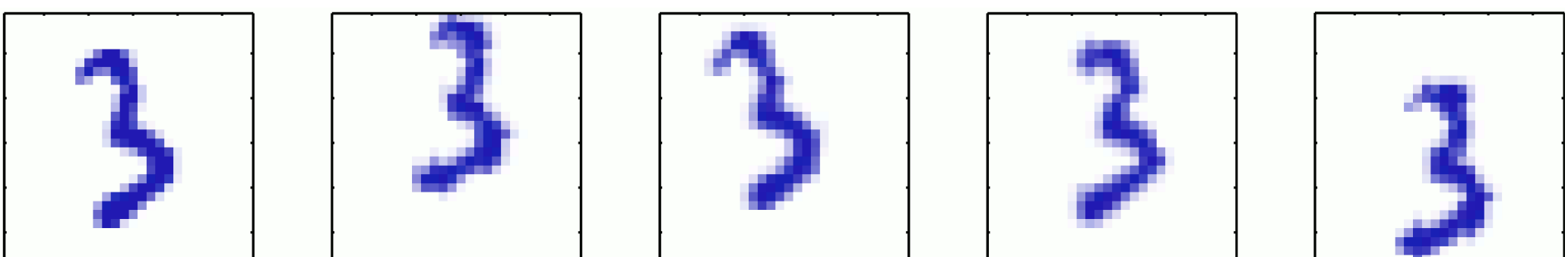


Figure 12.1 of Pattern Recognition and Machine Learning.

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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ 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 $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$.

We start by pointing out that 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:

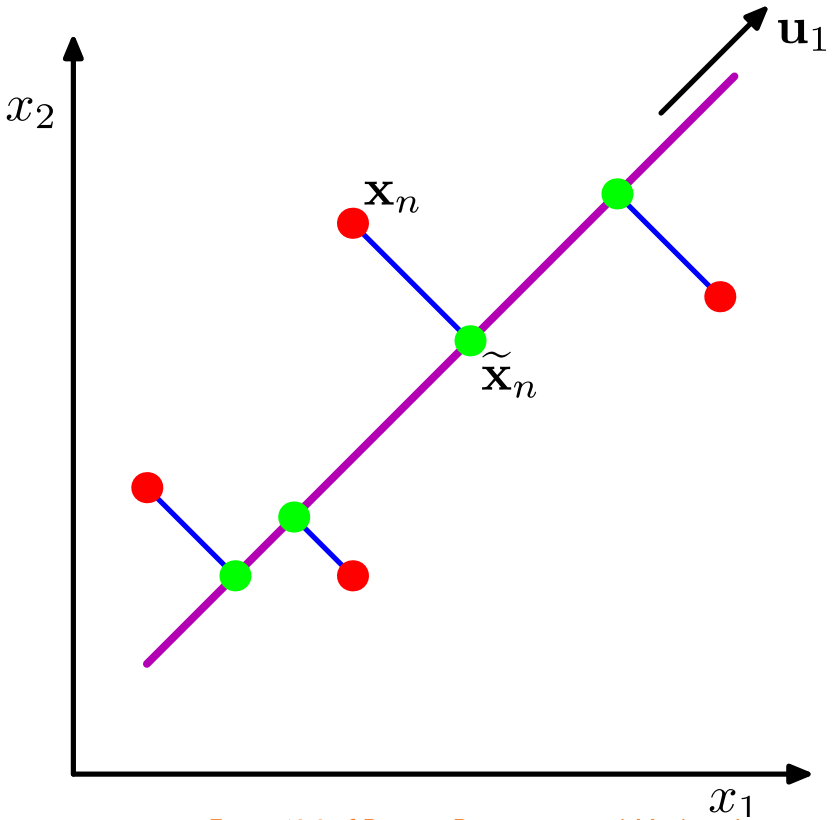


Figure 12.2 of Pattern Recognition and Machine Learning.

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

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

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

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

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

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

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.

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

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

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 will be 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, $\boldsymbol{u}_i^T \boldsymbol{u}_j = \left[i == j \right]$.

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 will approximate the data by the first M basis vectors:

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

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

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

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

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

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 eigenvectors of $D - M$ smallest eigenvalues.

PCA Applications – Data Compression

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

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

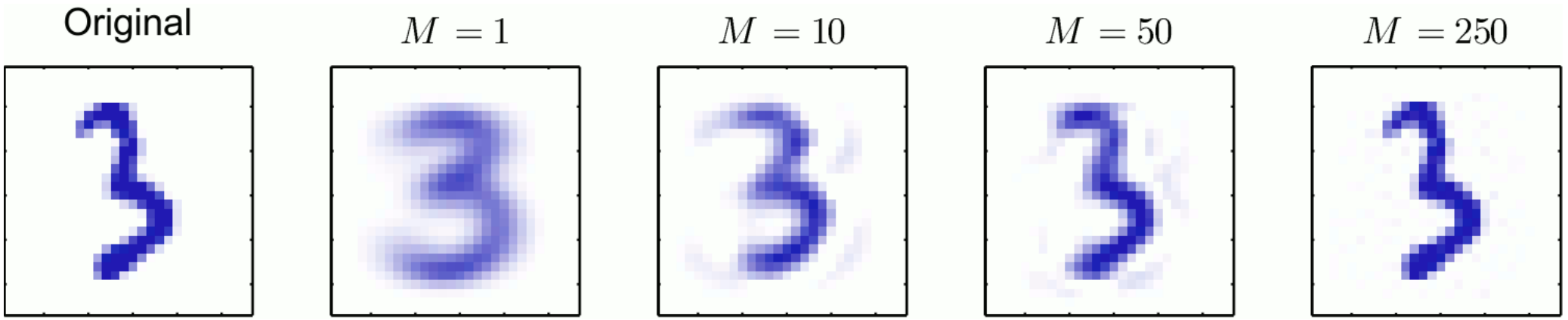


Figure 12.5 of Pattern Recognition and Machine Learning.

PCA Applications – Data Compression

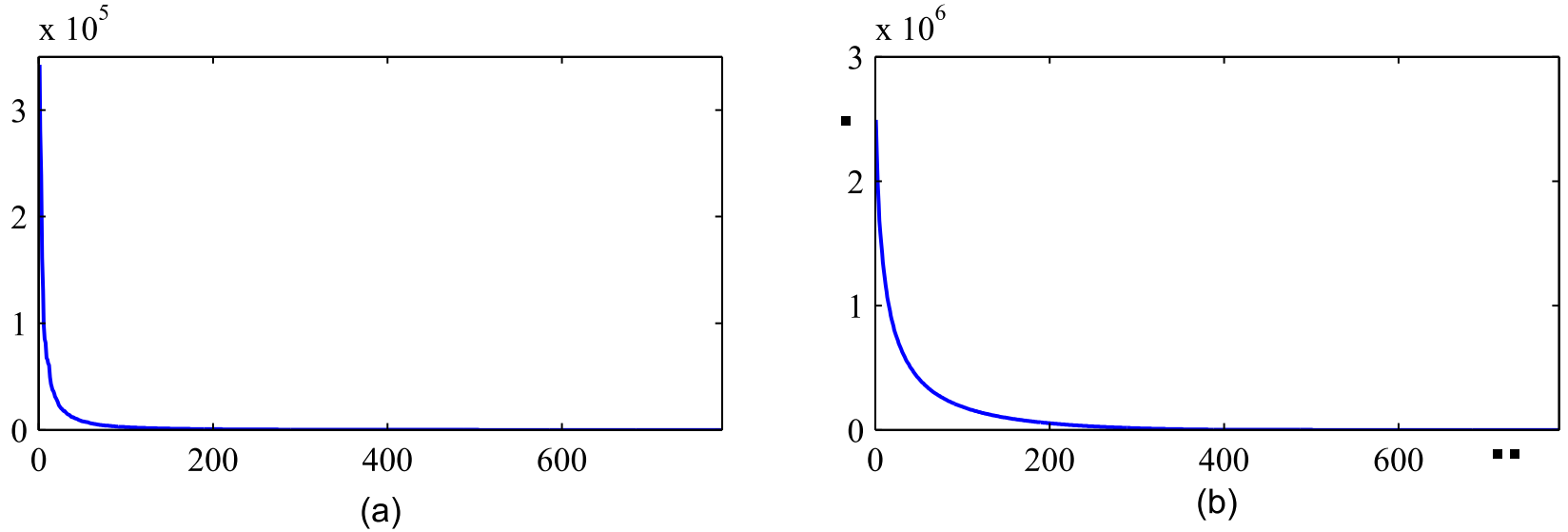


Figure 12.4 of Pattern Recognition and Machine Learning.

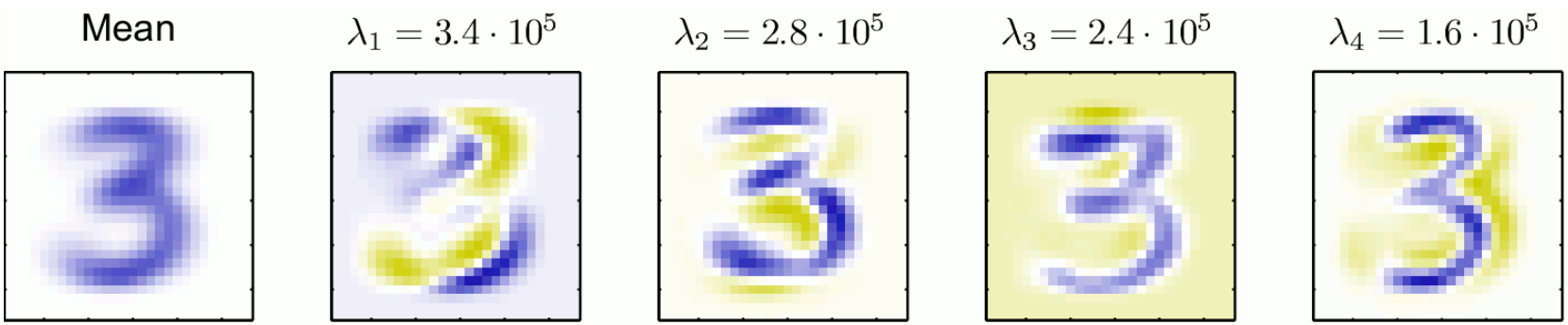


Figure 12.3 of Pattern Recognition and Machine Learning.

PCA Applications – Whitening or Sphering

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 $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{S}$ and $\boldsymbol{\Lambda}$ is the diagonal matrix of the corresponding eigenvalues (i.e., $\boldsymbol{SU} = \boldsymbol{U\Lambda}$), 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

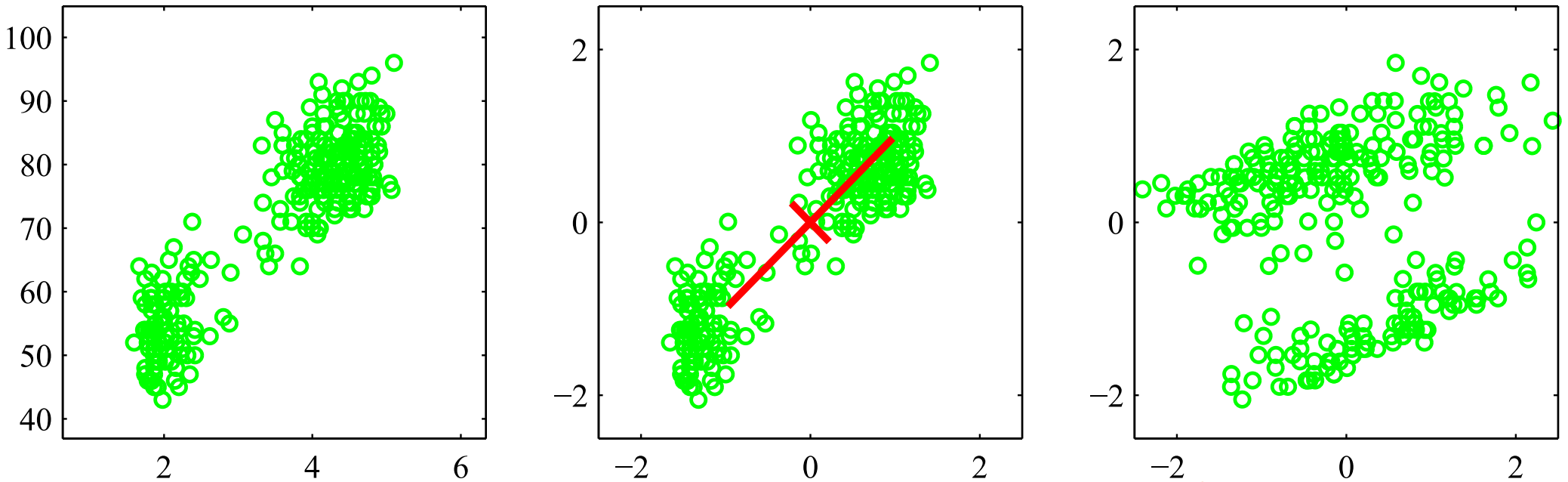


Figure 12.6 of Pattern Recognition and Machine Learning.

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.

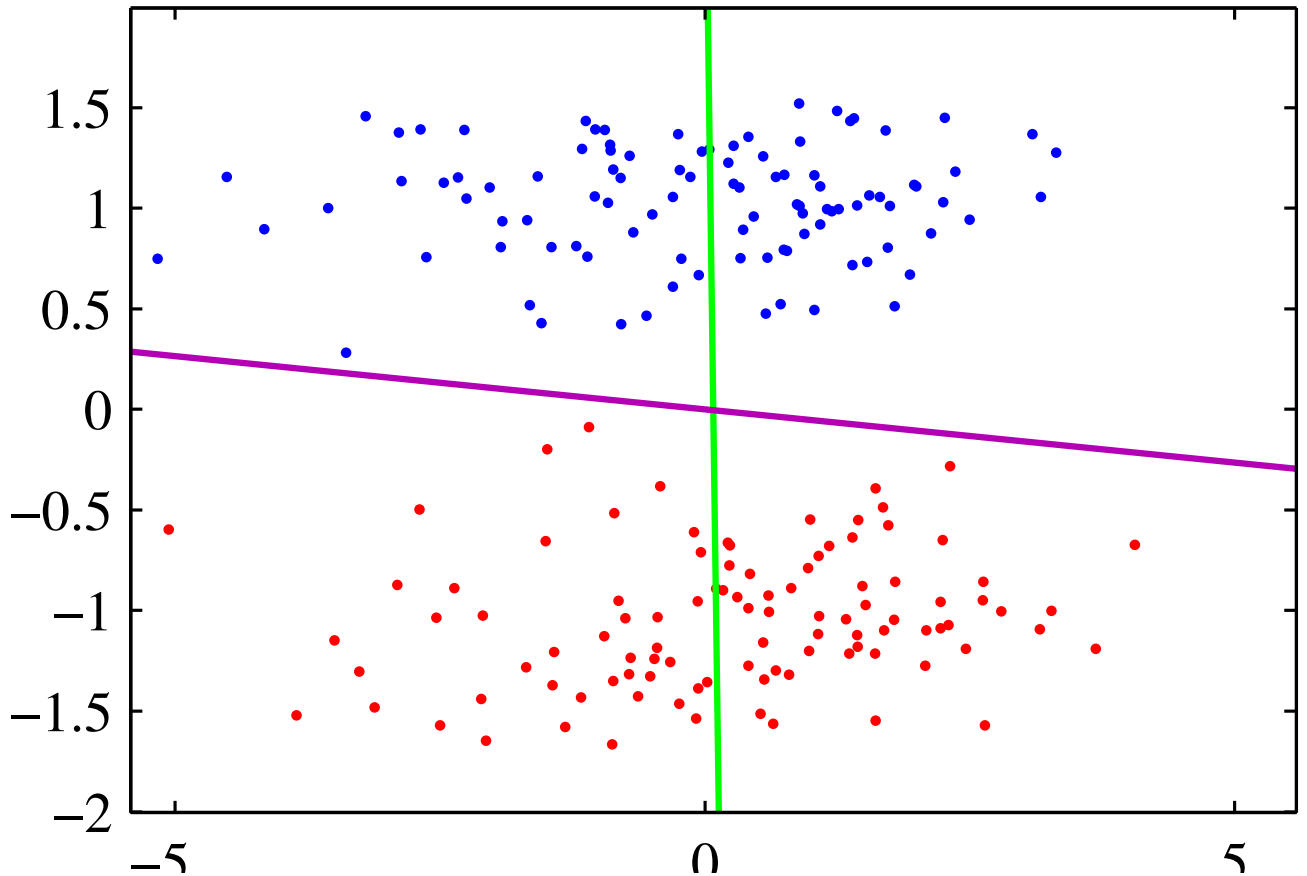


Figure 12.7 of Pattern Recognition and Machine Learning.

Principal Component Analysis and MLPs

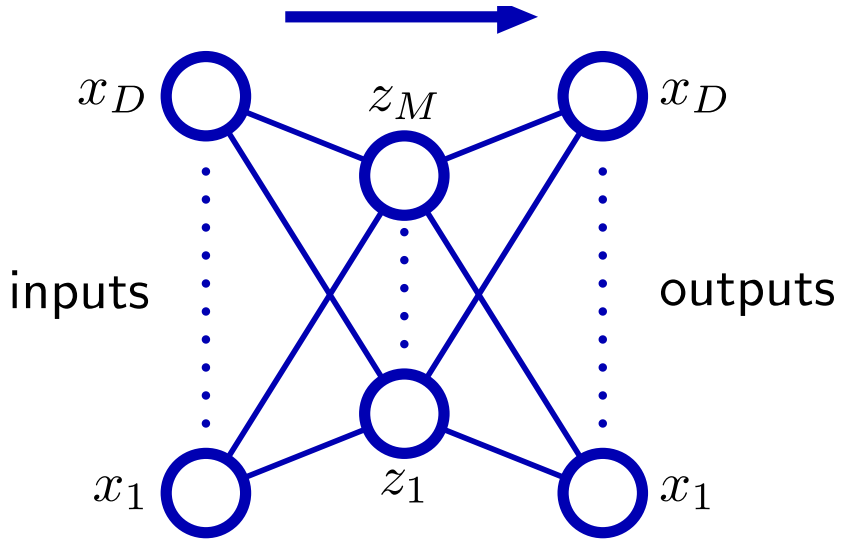


Figure 12.8 of Pattern Recognition and Machine Learning.

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.

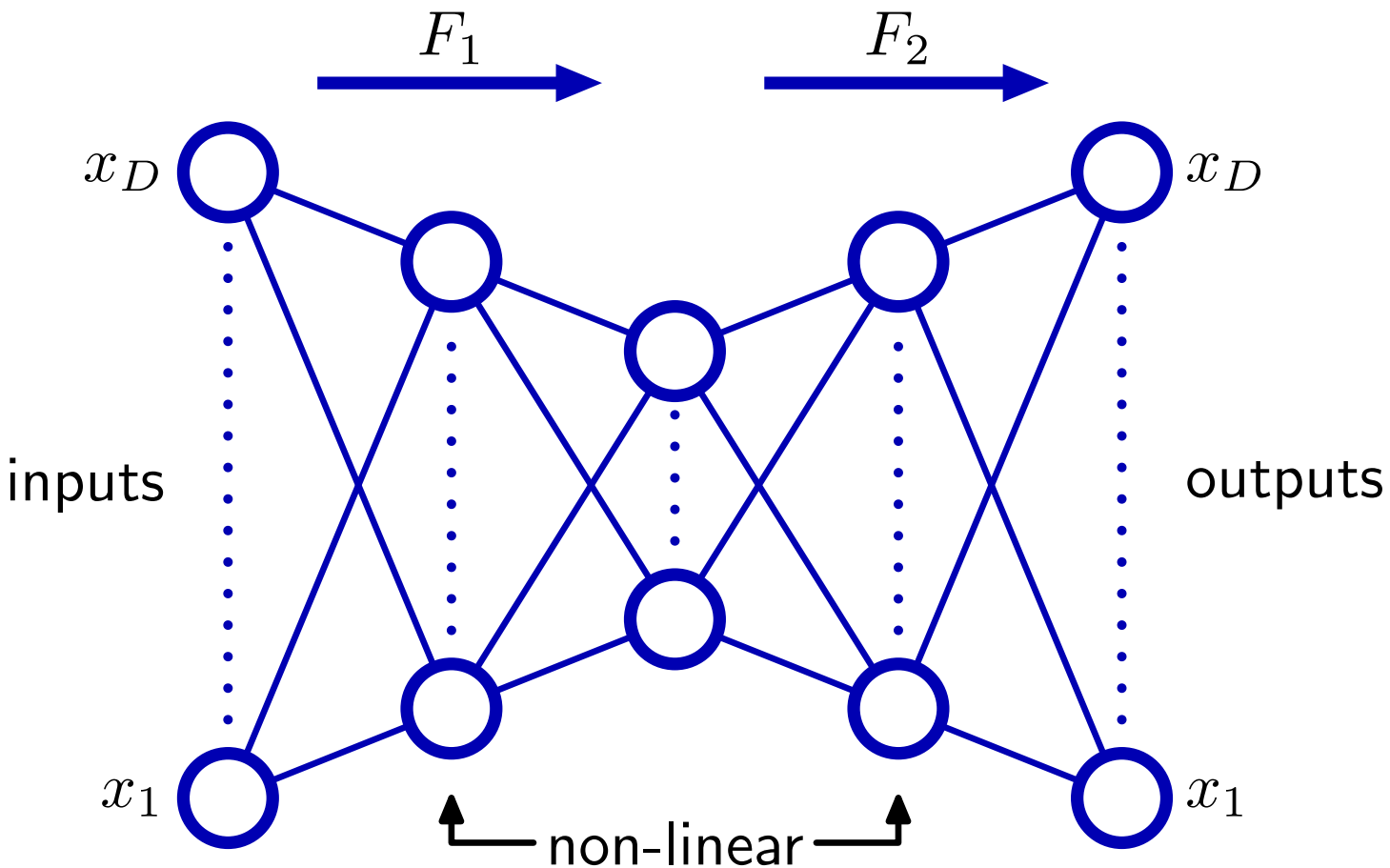


Figure 12.9 of Pattern Recognition and Machine Learning.

Computing PCA

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

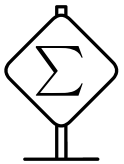
Input: Diagonalizable matrix $\boldsymbol{A}$ with a dominant eigenvalue.

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 non-zero first coordinate a_1 (which has probability very close to 1), then repeated multiplication with $\boldsymbol{A}$ will converge 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". Considering $\boldsymbol{A} - \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T$:

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

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

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

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

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 $S \leftarrow \frac{1}{N}\left(\boldsymbol{X} - \boldsymbol{\mu}\right)^T \left(\boldsymbol{X} - \boldsymbol{\mu}\right)$.
- 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 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).

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

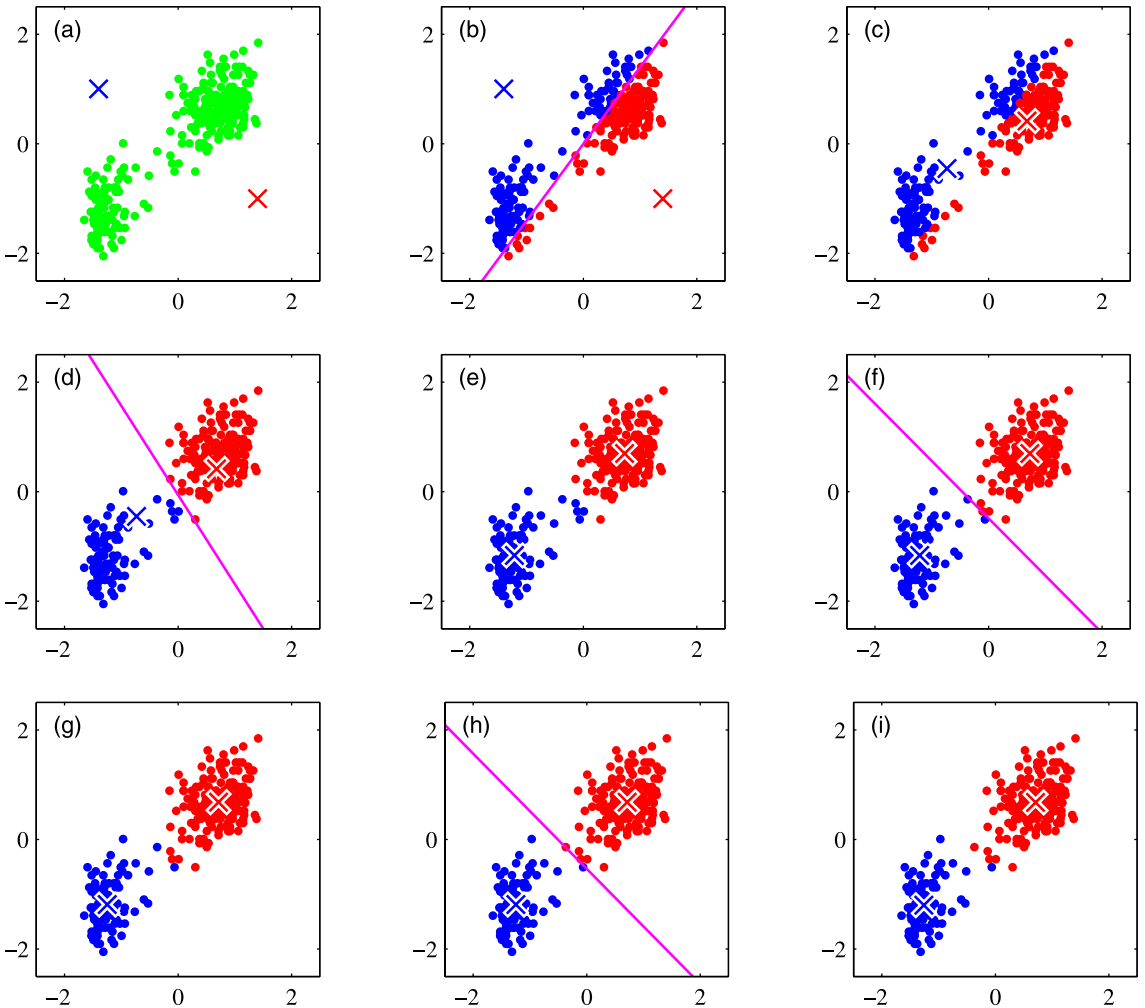


Figure 9.1 of Pattern Recognition and Machine Learning.

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.

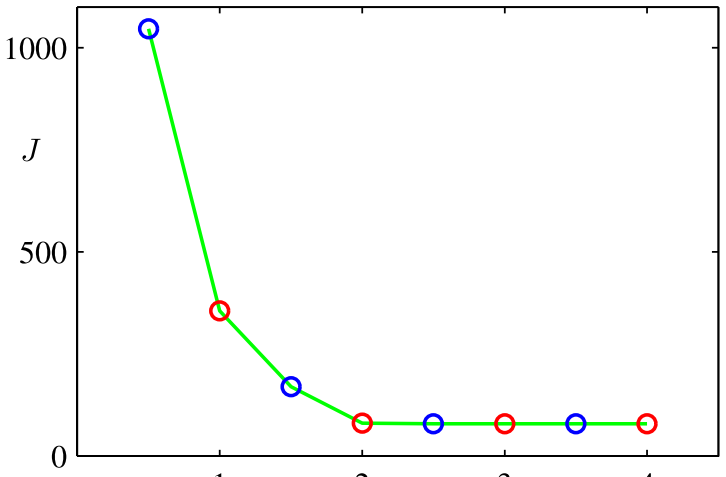


Figure 9.2 of Pattern Recognition and Machine Learning.

K-Means clustering therefore converges to a local optimum.
However, it is quite sensitive to the starting initialization:

- It is common practise 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 $\mathcal{O}(\log K)$ approximation ratio in expectation.

K-Means Clustering

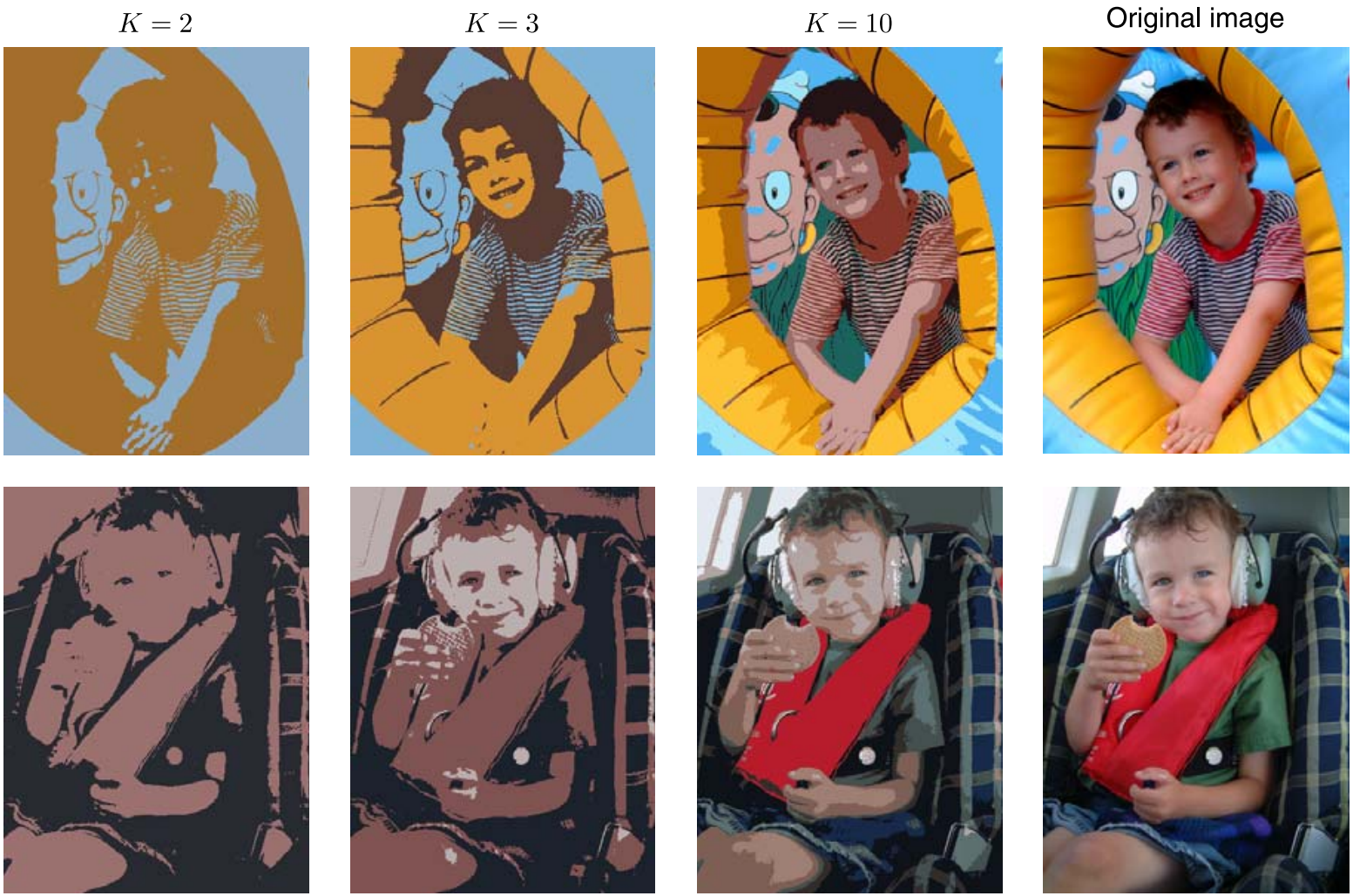
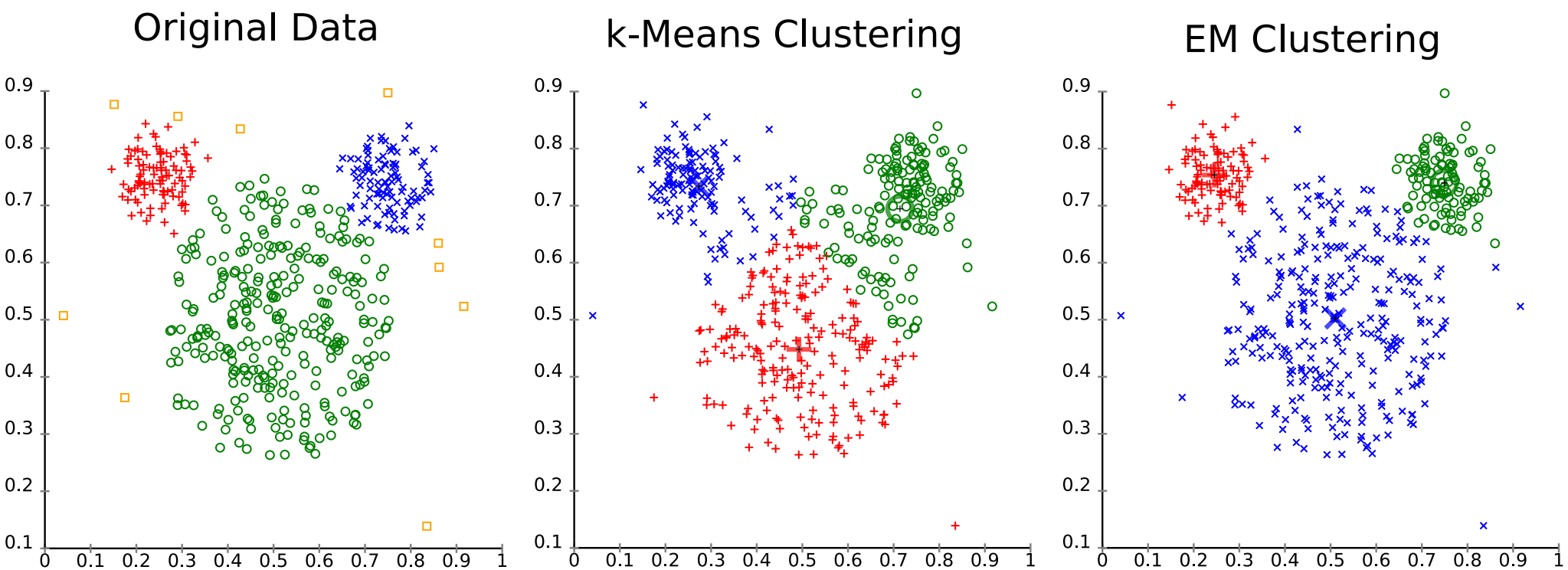


Figure 9.3 of Pattern Recognition and Machine Learning.

Gaussian Mixture vs K-Means

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

Different cluster analysis results on "mouse" data set:



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

Multivariate Gaussian Distribution

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 $\boldsymbol{x}$, the multivariate Gaussian distribution takes the form

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

The biggest difference compared to the single-dimensional case is the *covariance matrix* $\boldsymbol{\Sigma}$, which is (in the non-degenerate case, which is the only one considered here) a *symmetrical positive-definite matrix* of size $D \times D$.

Multivariate Gaussian Distribution

If the covariance matrix is an identity, then the multivariate Gaussian distribution simplifies to

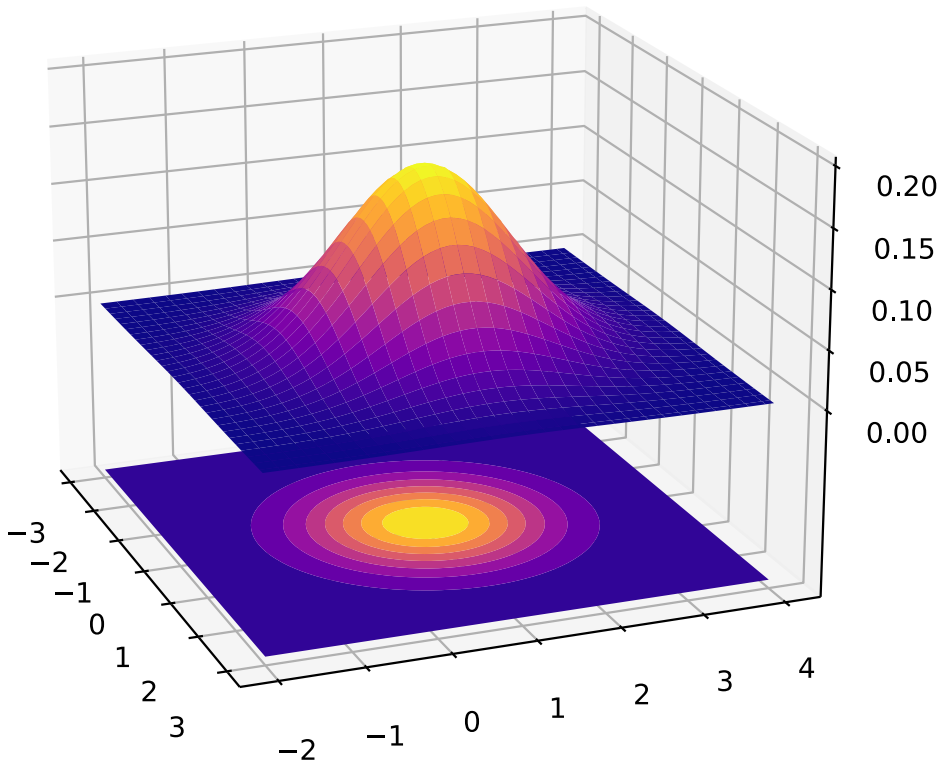
$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{I}) = \frac{1}{\sqrt{(2\pi)^D}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

We can rewrite the exponent in this case to

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

Therefore, the constant surfaces are concentric circles centered at the mean $\boldsymbol{\mu}$.

The same holds if the covariance is $\sigma^2\boldsymbol{I}$, only the circles' diameter changes.

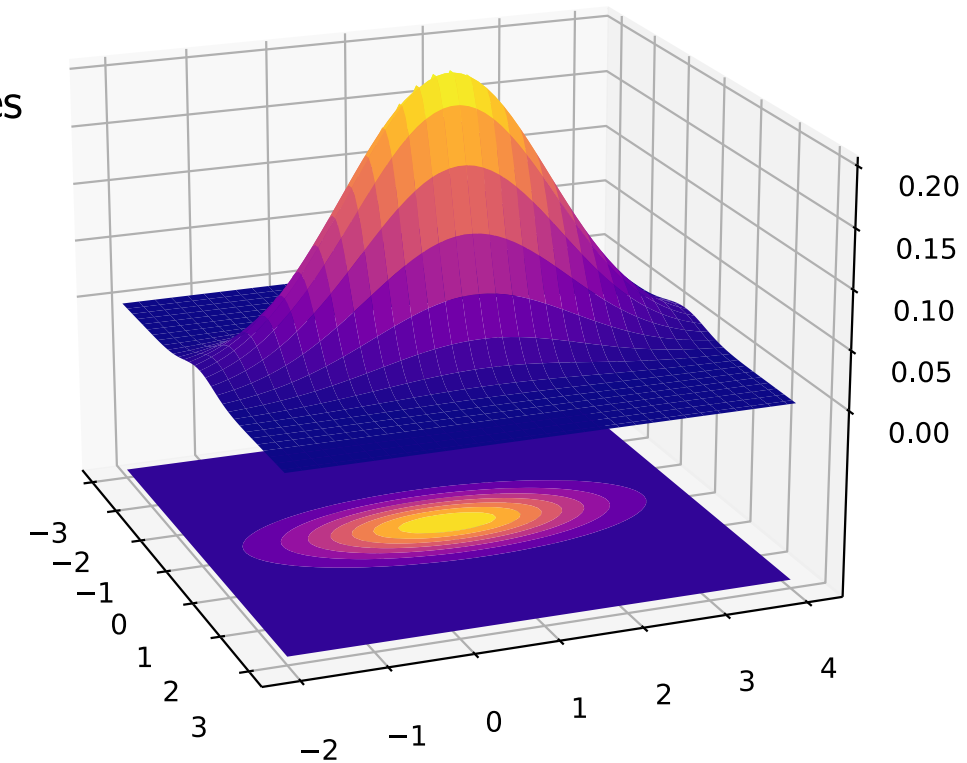


Multivariate Gaussian Distribution

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

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}) \propto \exp\left(-\sum_i \frac{1}{2\boldsymbol{\Lambda}_{i,i}}(\boldsymbol{x}_i - \boldsymbol{\mu}_i)^2\right).$$

The constant surfaces in this case are axis-aligned ellipses centered at the mean $\boldsymbol{\mu}$ with size of the axes depending on the corresponding diagonal entries in the covariance matrix.



Multivariate Gaussian Distribution

In the general case of a full covariance matrix, the fact that it is positive definite implies it has real positive *eigenvalues* λ_i. Considering the corresponding eigenvectors $\boldsymbol{u}_i$, it can be shown that the constant surfaces are again ellipses centered at $\boldsymbol{\mu}$, but this time rotated so that their axes are the eigenvectors $\boldsymbol{u}_i$ with sizes $\lambda_i^{1/2}$.

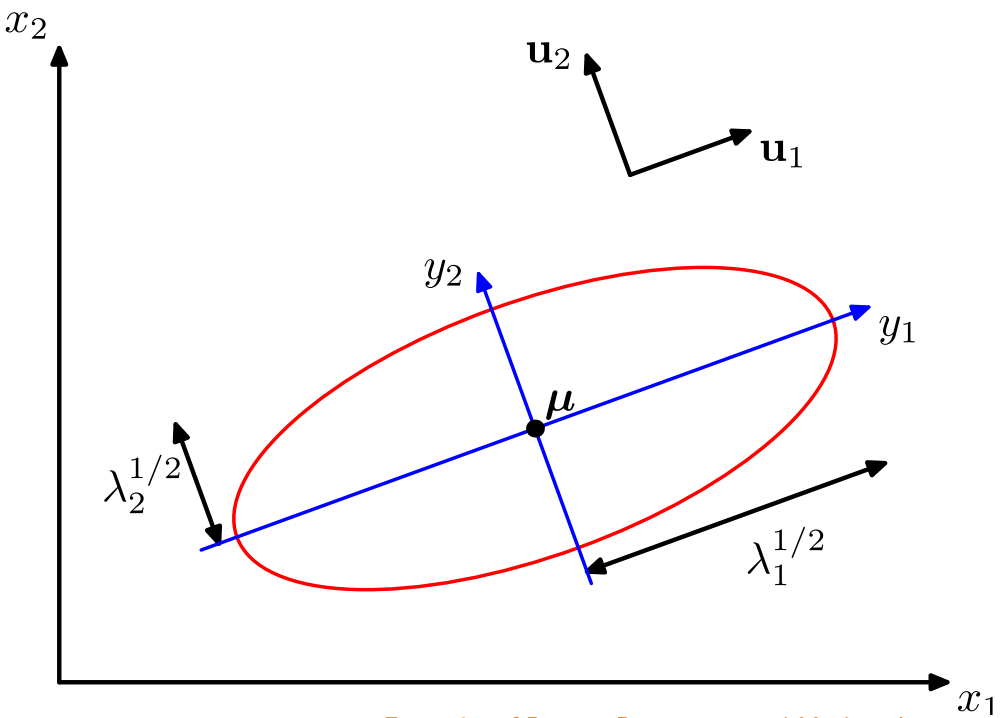
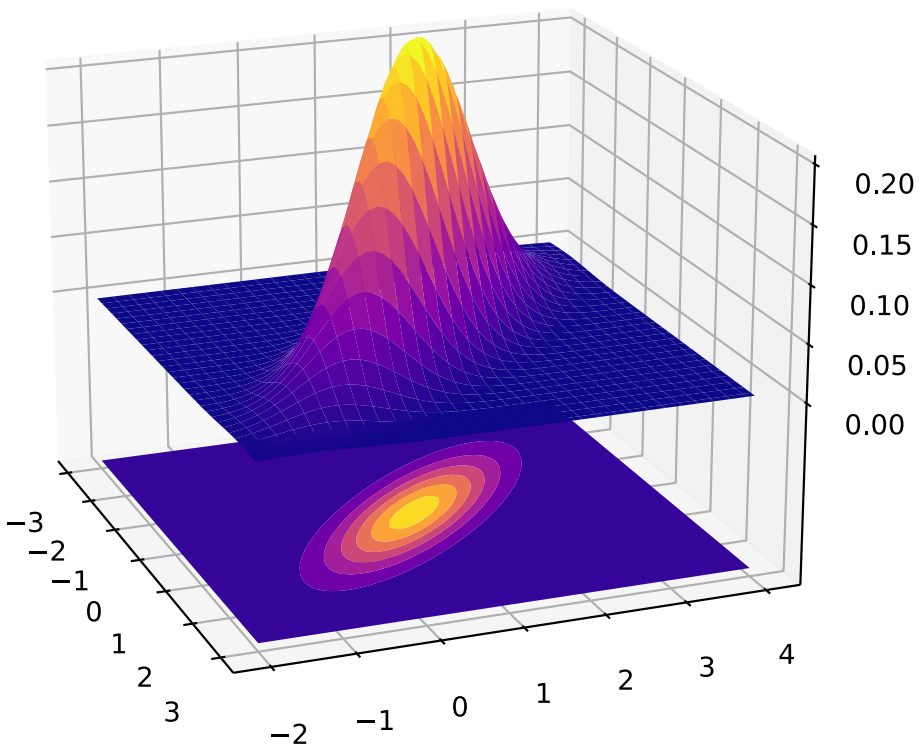


Figure 2.7 of Pattern Recognition and Machine Learning.



Multivariate Gaussian Distribution

Generally, we can rewrite a positive-definite matrix $\boldsymbol{\Sigma}$ as $(\boldsymbol{U}\boldsymbol{\Lambda}^{1/2})(\boldsymbol{U}\boldsymbol{\Lambda}^{1/2})^T$, and then

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

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

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

- the $\sigma^2\boldsymbol{I}$ has a single parameter,
- the $\boldsymbol{\Lambda}$ has D parameters,
- the full covariance matrix $\boldsymbol{\Sigma}$ has $\binom{D+1}{2}$ parameters, i.e., $\Theta(D^2)$.

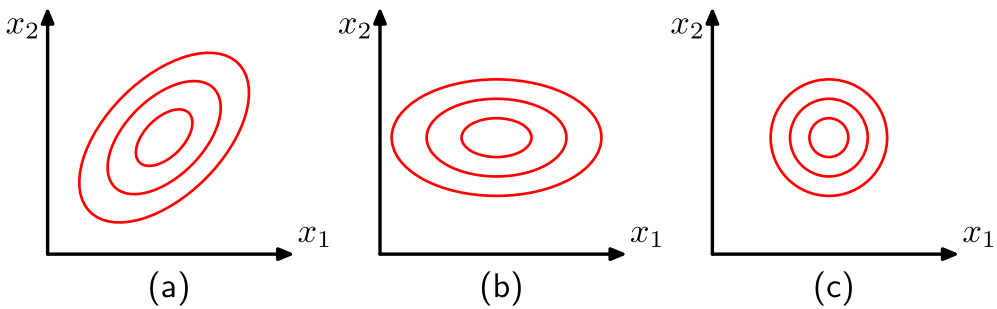


Figure 2.8 of Pattern Recognition and Machine Learning.

Gaussian Mixture

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.

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

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Therefore, each cluster is parametrized as $\mathcal{N}(\boldsymbol{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Gaussian Mixture

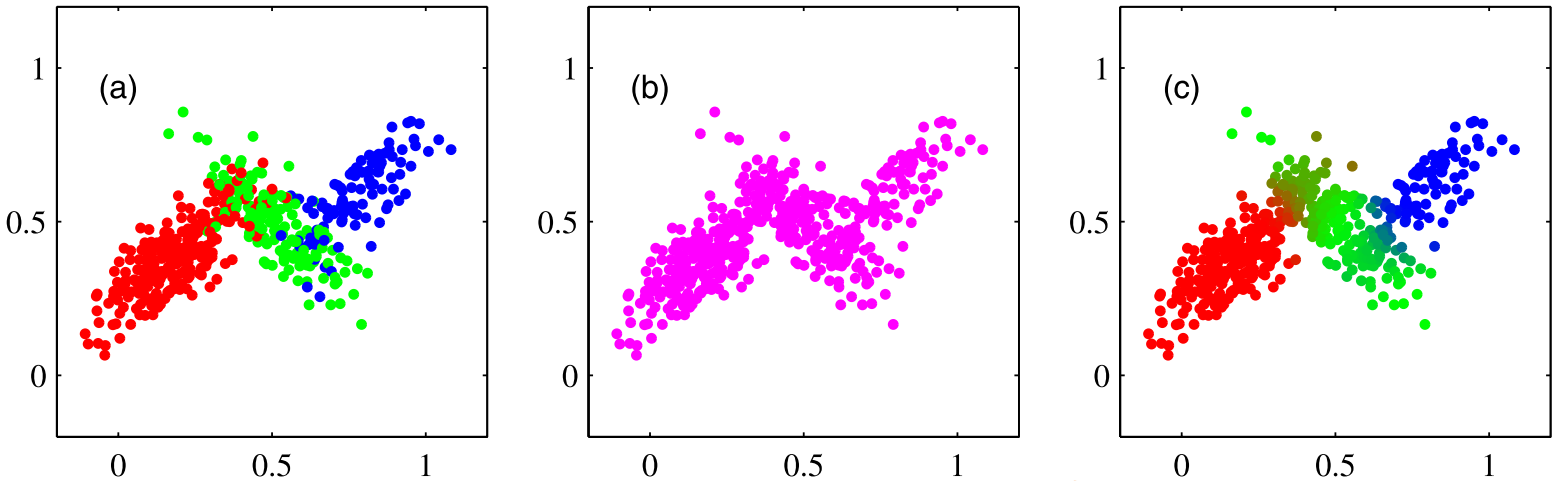


Figure 9.5 of Pattern Recognition and Machine Learning.

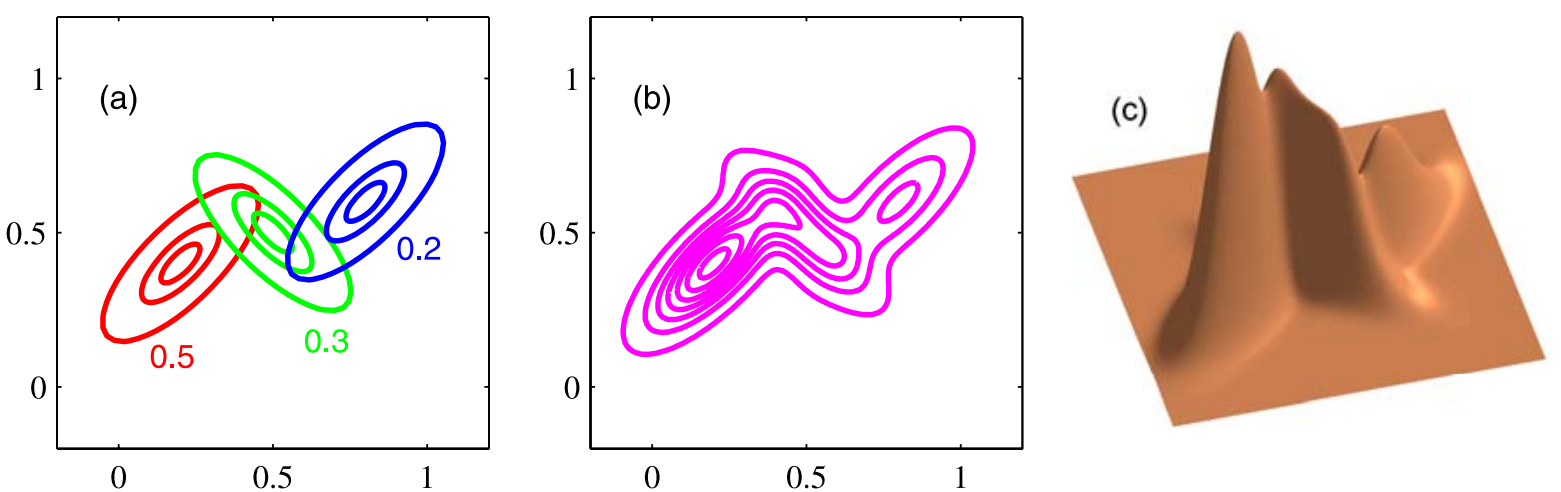


Figure 2.23 of Pattern Recognition and Machine Learning.