SVD, PCA, K-Means

Jindřich Libovický (reusing some materials by Milan Straka)

December 8, 2024

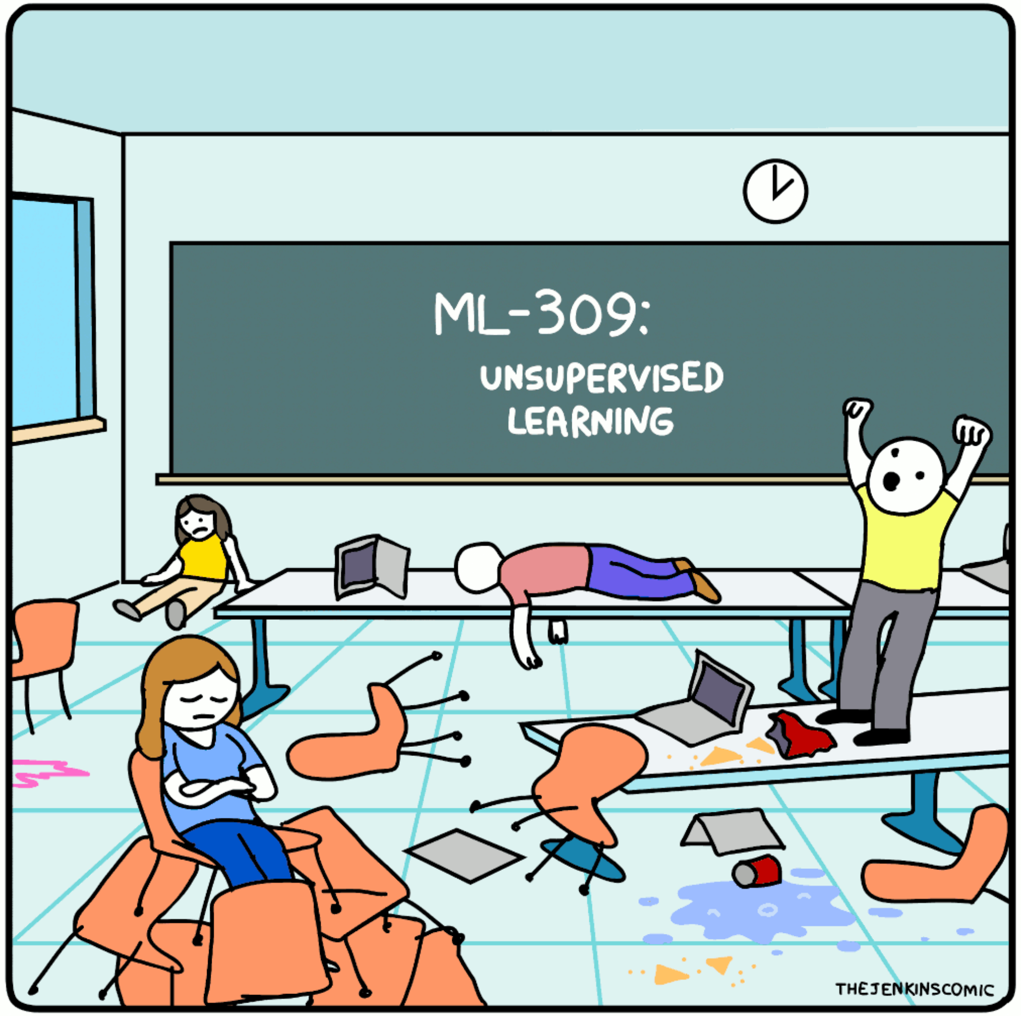
Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Today's Lecture Objectives

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 and data visualization
- Implement the k-means algorithm and use it for clustering

Unsupervised Machine Learning



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

Recap of Linear Algebra

Matrix Decompositions

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{B}$$

- $\boldsymbol{B}$ tell us how to construct $\boldsymbol{X}$ using columns of $\boldsymbol{A}$
- $\boldsymbol{A}$ tell us how to construct $\boldsymbol{X}$ using rows of $\boldsymbol{B}$

$\boldsymbol{X}$ 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

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*.

Quick (almost) proof of orthogonality

$\boldsymbol{A}\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1$, we transpose both sides and get $(\boldsymbol{A}\boldsymbol{v}_1)^T = \lambda_1 \boldsymbol{v}_1^T$

Multiply by $\boldsymbol{v}_2$ from right, we get $\boldsymbol{v}_1^T \boldsymbol{A}^T \boldsymbol{v}_2 = \lambda_1 \boldsymbol{v}_1^T \boldsymbol{v}_2$, but $\boldsymbol{A}$ is symmetric and $\boldsymbol{A}^T = \boldsymbol{A}$, so $\boldsymbol{v}_1^T(\boldsymbol{A}\boldsymbol{v}_2) = \lambda_2 \boldsymbol{v}_1^T \boldsymbol{v}_2 \Rightarrow \lambda_2 \boldsymbol{v}_1^T \boldsymbol{v}_2 = \lambda_1 \boldsymbol{v}_1^T \boldsymbol{v}_2$, resulting in

$$(\lambda_2 - \lambda_1)(\boldsymbol{v}_1^T \boldsymbol{v}_2) = 0.$$

Linear Algebra Refresh – Eigenvalue Decomposition

We can express real symmetric $\boldsymbol{A}$ using the **eigenvalue decomposition**

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

where:

- $\boldsymbol{V}$ is a matrix, whose columns are the orthonormal 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.

Quick derivation

$$\boldsymbol{A}\boldsymbol{V} = \boldsymbol{\Lambda}\boldsymbol{V} = \boldsymbol{V}\boldsymbol{\Lambda}$$

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

Because $\boldsymbol{V}$ is orthonormal $\boldsymbol{V}^T = \boldsymbol{V}^{-1}$, so $\boldsymbol{V}\boldsymbol{V}^T = 1$ and $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$.

Singular Value Decomposition

Singular Value Decomposition

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

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

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

$$\boldsymbol{X}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{\Sigma} \qquad \Rightarrow \qquad \boldsymbol{X}\boldsymbol{v}_k = \sigma_k \boldsymbol{u}_k \quad \forall k = 1, \ldots, r$$

$$\boldsymbol{X} \begin{bmatrix} & \vdots & \\ \boldsymbol{v}_1 \cdots & \boldsymbol{v}_r & \cdots \boldsymbol{v}_n \\ & \vdots & \end{bmatrix} = \begin{bmatrix} & \vdots & \\ \boldsymbol{u}_1 \cdots & \boldsymbol{u}_r & \cdots \boldsymbol{u}_m \\ & \vdots & \end{bmatrix} \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & \mathbf{0} \\ & & \sigma_r & \\ \hline & \mathbf{0} & & 0 \end{array}\right]$$

SVD: Proof

Assuming SVD exists, we can write $\boldsymbol{U}$ and $\boldsymbol{V}$ as eigenvector decomposition of row and column similarities:

$$\boldsymbol{X}\boldsymbol{X}^T = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T = \boldsymbol{U}\boldsymbol{\Sigma}(\boldsymbol{V}^T\boldsymbol{V})\boldsymbol{\Sigma}^T\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T\boldsymbol{U}^T$$

$$\boldsymbol{X}^T\boldsymbol{X} = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T) = \boldsymbol{V}\boldsymbol{\Sigma}(\boldsymbol{U}^T\boldsymbol{U})\boldsymbol{\Sigma}^T\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\boldsymbol{V}^T$$

Let's take $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r]$ orthonormal eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$ and set $\sigma_k = \sqrt{\lambda_k}$, then, $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{v}_k = \sigma_k^2\boldsymbol{v}_k$. ($\lambda_k \geq 0$ because $\boldsymbol{X}\boldsymbol{X}^T$ is positive semi-definite.)

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

$$\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{u}_k = \boldsymbol{X}\boldsymbol{X}^T\underbrace{\left(\frac{\boldsymbol{X}\boldsymbol{v}_k}{\sigma_k}\right)}_{\text{def. of } \boldsymbol{u}_k} = \boldsymbol{X}\underbrace{\left(\frac{\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{v}_k}{\sigma_k}\right)}_{\boldsymbol{v}_k \text{ is eigenvector of } \boldsymbol{X}^T\boldsymbol{X}} = \boldsymbol{X}\frac{\sigma_k^2\boldsymbol{v}_k}{\sigma_k} = \sigma_k^2\underbrace{\left(\frac{\boldsymbol{X}\boldsymbol{v}_k}{\sigma_k}\right)}_{\text{def. of } \boldsymbol{u}_k} = \sigma_k^2\boldsymbol{u}_k$$

Interpretation of SVD

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

$$\boldsymbol{X} = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^T + \ldots + \sigma_r \boldsymbol{u}_r \boldsymbol{v}_r^T$$

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

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

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

$\boldsymbol{X} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{X}_k = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \ldots + \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T$ its approximation using SVD. For each $\boldsymbol{B} \in \mathbb{R}^{n \times m}$ of rank k

$$||\boldsymbol{X} - \boldsymbol{X}_k||_F \leq ||\boldsymbol{X} - \boldsymbol{B}||_F.$$

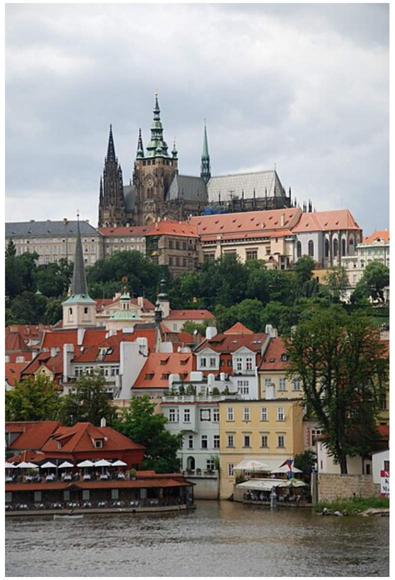
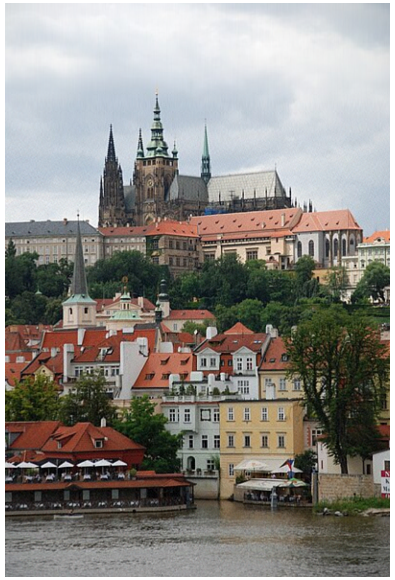
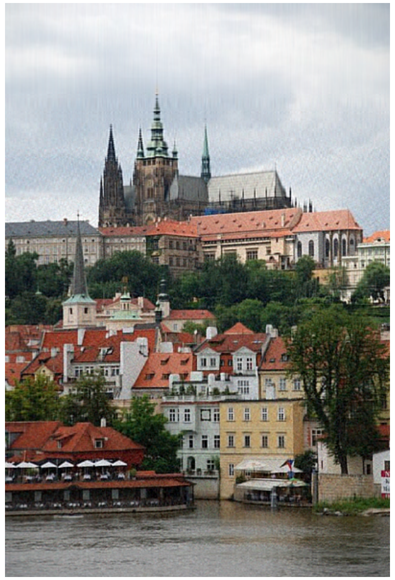
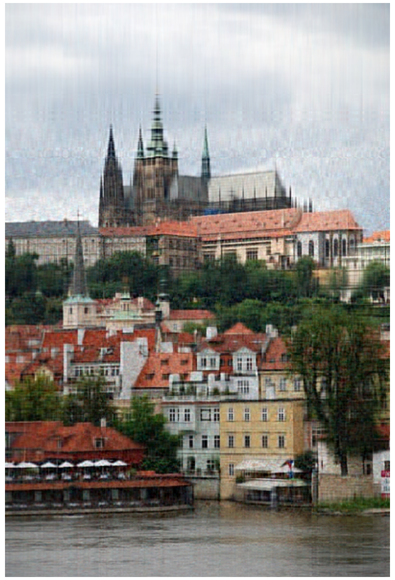
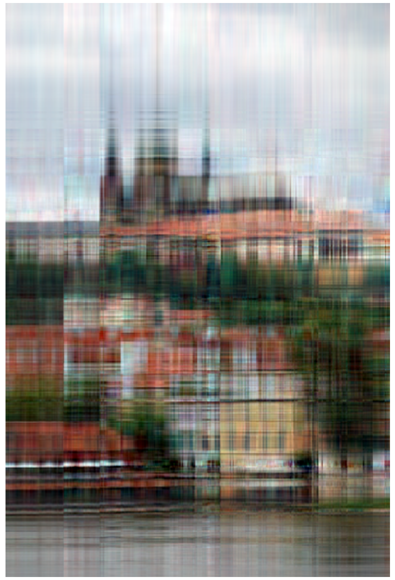
Argument why it is a good idea

- $||\boldsymbol{X}||_F = \sqrt{\sum_i^n \sum_j^m x_{ij}^2} = \sqrt{\mathrm{trace}(\boldsymbol{X}^T\boldsymbol{X})}$ (trace is the sum of the diagonal)
- Multiplying $\boldsymbol{A}$ by an orthonormal matrix $\boldsymbol{U}$ does not change the norm of $\boldsymbol{A}$:

$$||\boldsymbol{U}\boldsymbol{A}||_F^2 = \mathrm{trace}((\boldsymbol{U}\boldsymbol{A})^T\boldsymbol{U}\boldsymbol{A}) = \mathrm{trace}(\boldsymbol{A}^T \underbrace{\boldsymbol{U}^T\boldsymbol{U}}_{\boldsymbol{I}} \boldsymbol{A}) = \mathrm{trace}(\boldsymbol{A}^T\boldsymbol{A}) = ||\boldsymbol{A}||_F^2.$$

- The Frobenius norm of $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ is the L^2 norm of the diagonal of $\boldsymbol{\Sigma}$.
- The best strategy to keep the most of the norm is removing the smallest values.

Image Compression using SVD

400 components	200 components	100 components	50 components	10 components
				
actually bigger	1.2× smaller	2.4× smaller	4.8× smaller	24× smaller

SVD in Recommender Systems

- We 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 users 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”: Can be used for **similarity search**.
- In practice, slightly modified versions of SVD are used.

Principal Component Analysis

From SVD to Principal Component Analysis

So far, SVD had geometric interpretation, let's add statistical interpretation.

When we apply SVD on mean-centered data $\boldsymbol{X} - \boldsymbol{\mu}$ (where $\boldsymbol{\mu}$ is row-wise average of $\boldsymbol{X}$), singular values get a new interpretation: **components explaining variability in the data**.

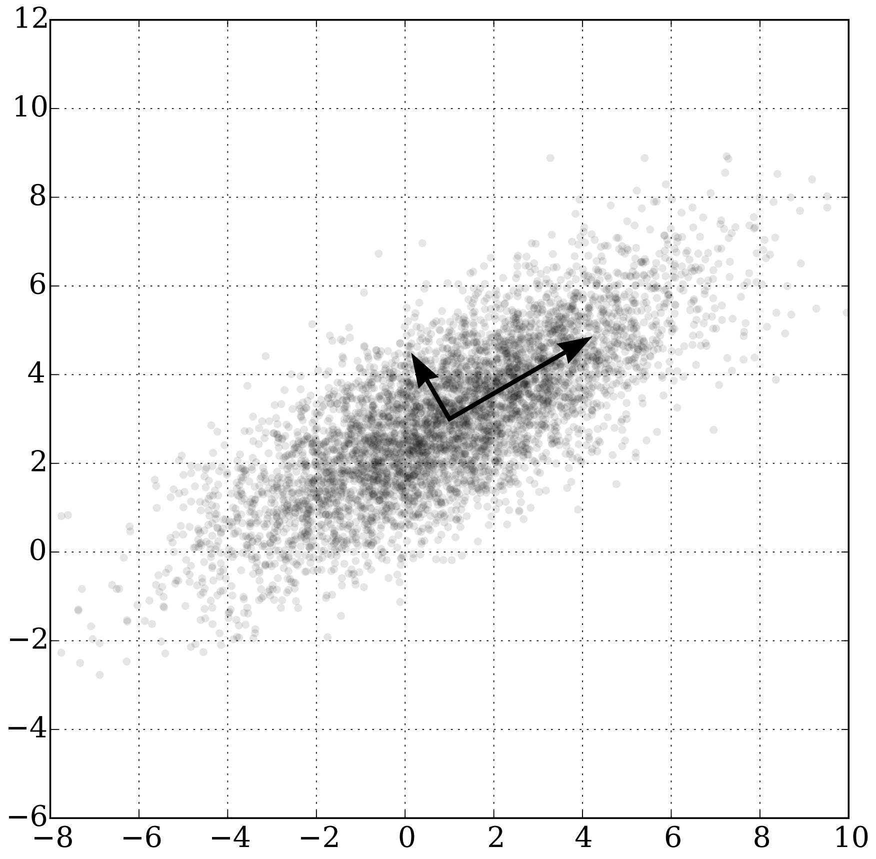
$$||\boldsymbol{X} - \boldsymbol{x}||_F^2 = \operatorname{trace} \underbrace{\left((\boldsymbol{X} - \boldsymbol{\mu})^T (\boldsymbol{X} - \boldsymbol{\mu})\right)}_{N \operatorname{Cov}(\boldsymbol{X})} = N \sum_i^D \operatorname{Var}(\boldsymbol{X}_{:,i})$$

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

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

Note the $\frac{1}{N}$ term scales down the eigenvalues compared to SVD, but keeps the eigenvectors unchanged.

Plot Example



Principle components in data sampled from a 2D Gaussian. $\boldsymbol{U\Sigma V}$ is SVD decomposition of mean-centered data matrix $\boldsymbol{X}$.

- Vectors of $(\boldsymbol{\Sigma V})$ define the directions of the components, so called **loadings**.
- Each vector of $\boldsymbol{U}$ represents one corresponding centered example $\boldsymbol{X} \in \boldsymbol{X} - \bar{\boldsymbol{x}}$ 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,
- data visualization.

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

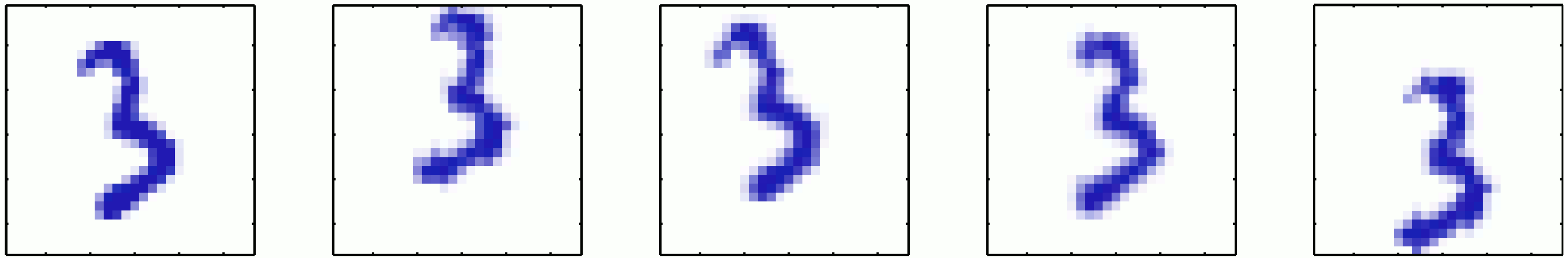
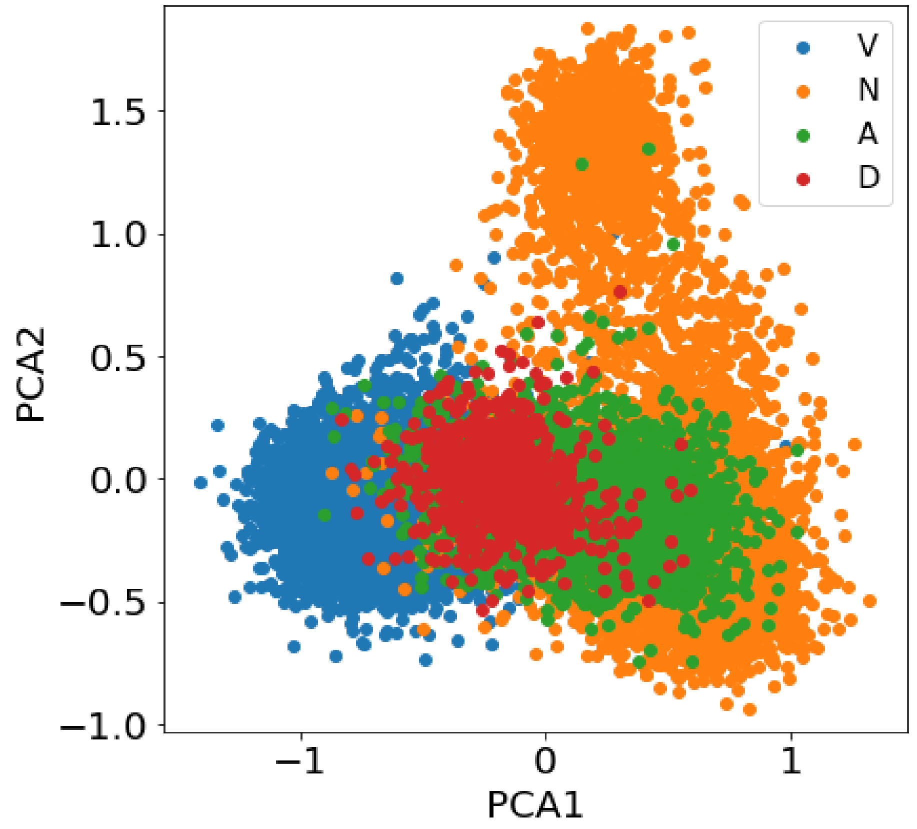


Figure 12.1 of Pattern Recognition and Machine Learning.

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*.

Data Visualization

Word embeddings from neural machine translation.



Marecek, D., Libovický, J., Musil, T., Rosa, R., & Limisiewicz, T. (2020). Hidden in the Layers: Interpretation of Neural Networks for Natural Language Processing. ISBN: 978-80-88132-10-3. Figure 4.

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. PCA = projecting on the magenta line, which does not help the classification.

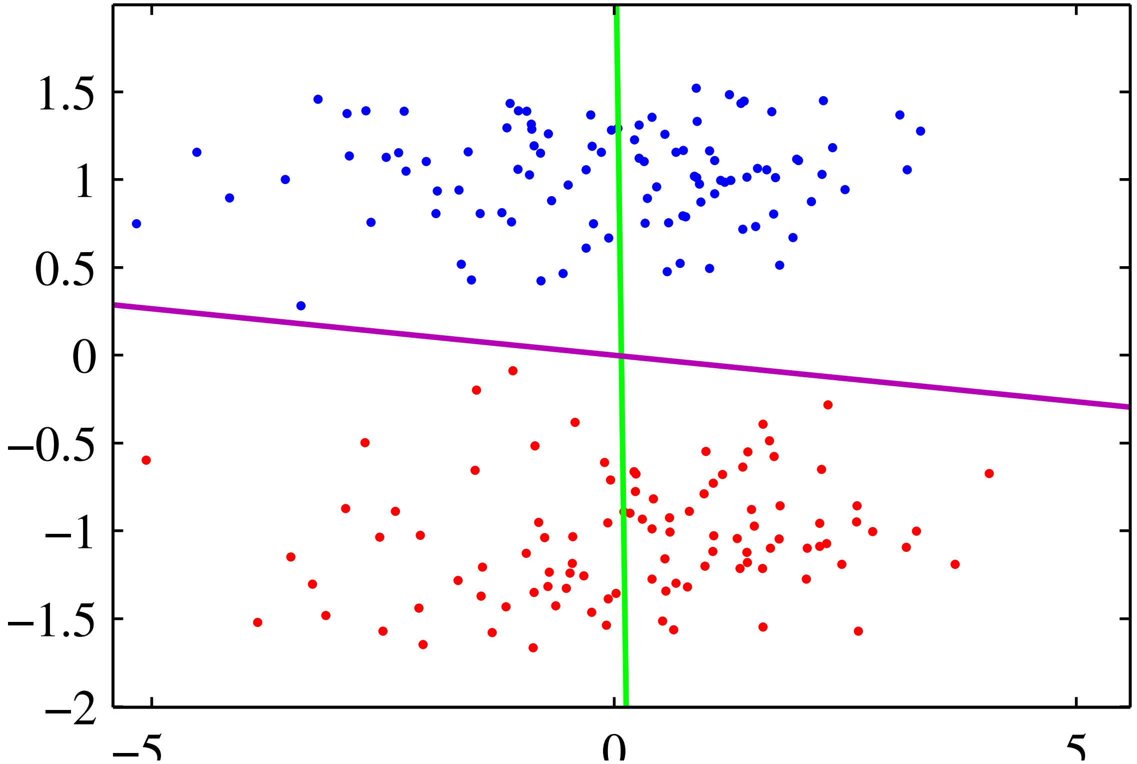


Figure 12.7 of Pattern Recognition and Machine Learning.

Power Iteration Algorithm

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 $a_1\boldsymbol{u}_1 + a_2\boldsymbol{u}_1 + a_3\boldsymbol{u}_1 \ldots$, then $\boldsymbol{A}\boldsymbol{v}$ is in the basis of the eigenvectors:

$$\boldsymbol{A}\boldsymbol{v} = \lambda_1 a_1 \boldsymbol{u}_1 + \lambda_2 a_2 \boldsymbol{u}_2 + \lambda_3 a_3 \boldsymbol{u}_3 \ldots$$

$$\boldsymbol{A}^k\boldsymbol{v} = \lambda_1^k a_1 \boldsymbol{u}_2 + \lambda_2^k a_2 \boldsymbol{u}_2 + \lambda_3^k a_3 \boldsymbol{u}_3 \ldots = \lambda_1^k \left(a_1\boldsymbol{u}_1 + \frac{\lambda_2^k}{\lambda_1^k} a_2 \boldsymbol{u}_2 + \frac{\lambda_3^k}{\lambda_1^k} a_3 \boldsymbol{u}_3 + \ldots \right)$$

Coordinates $\frac{\lambda_i^k}{\lambda_1^k}$ go to zero with $k \to \infty$. Normalization during the algorithm prevents the λ_1^k term from exploding.

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 $(\boldsymbol{X}-\boldsymbol{\mu})\boldsymbol{V}$, where the columns of $\boldsymbol{V}$ are $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_M$.
```

K-Means Clustering

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

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

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

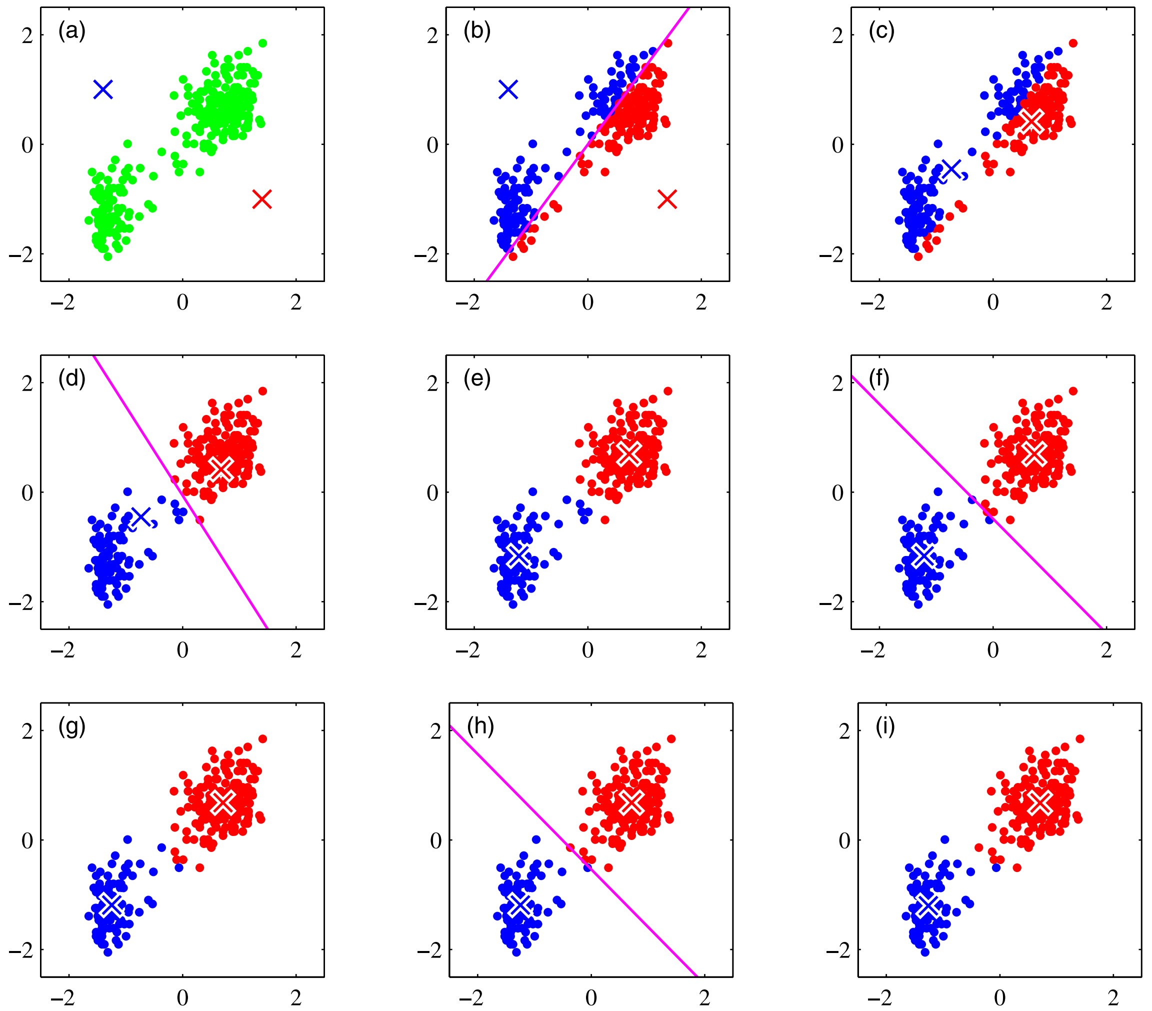


Figure 9.1 of Pattern Recognition and Machine Learning.

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.

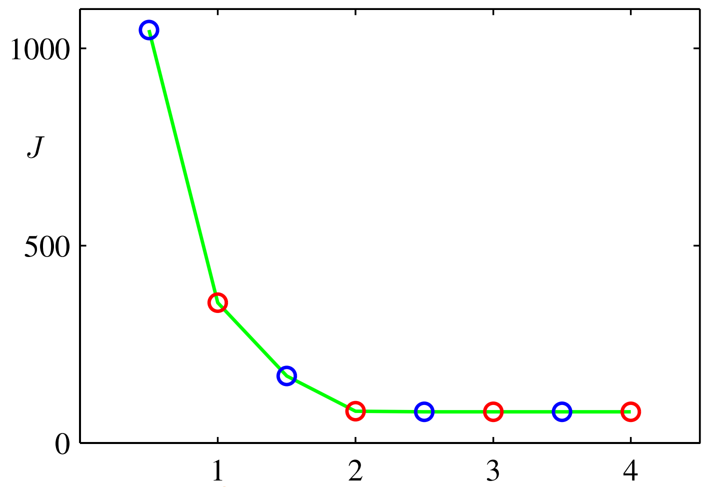


Figure 9.2 of Pattern Recognition and Machine Learning.

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

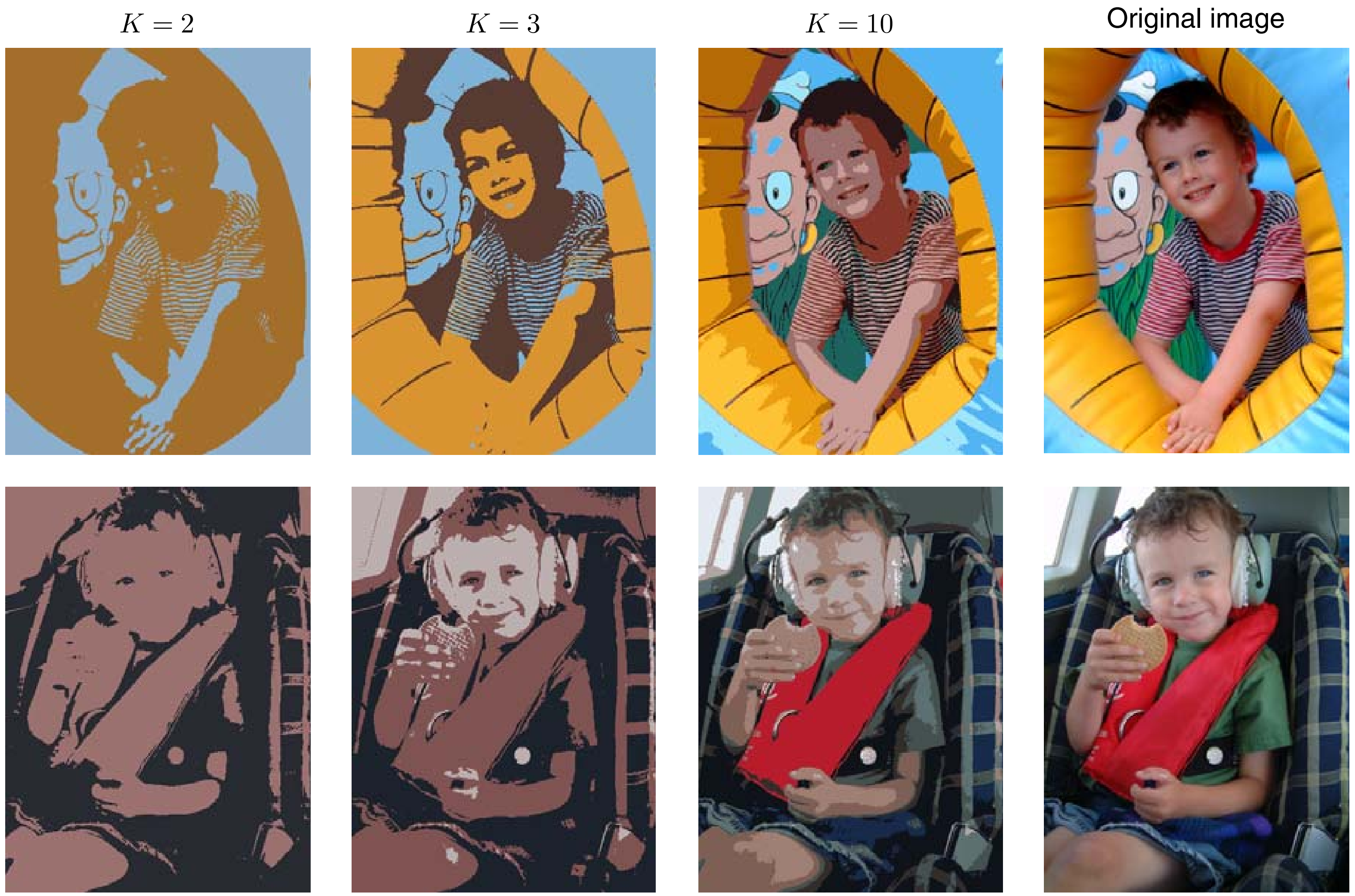
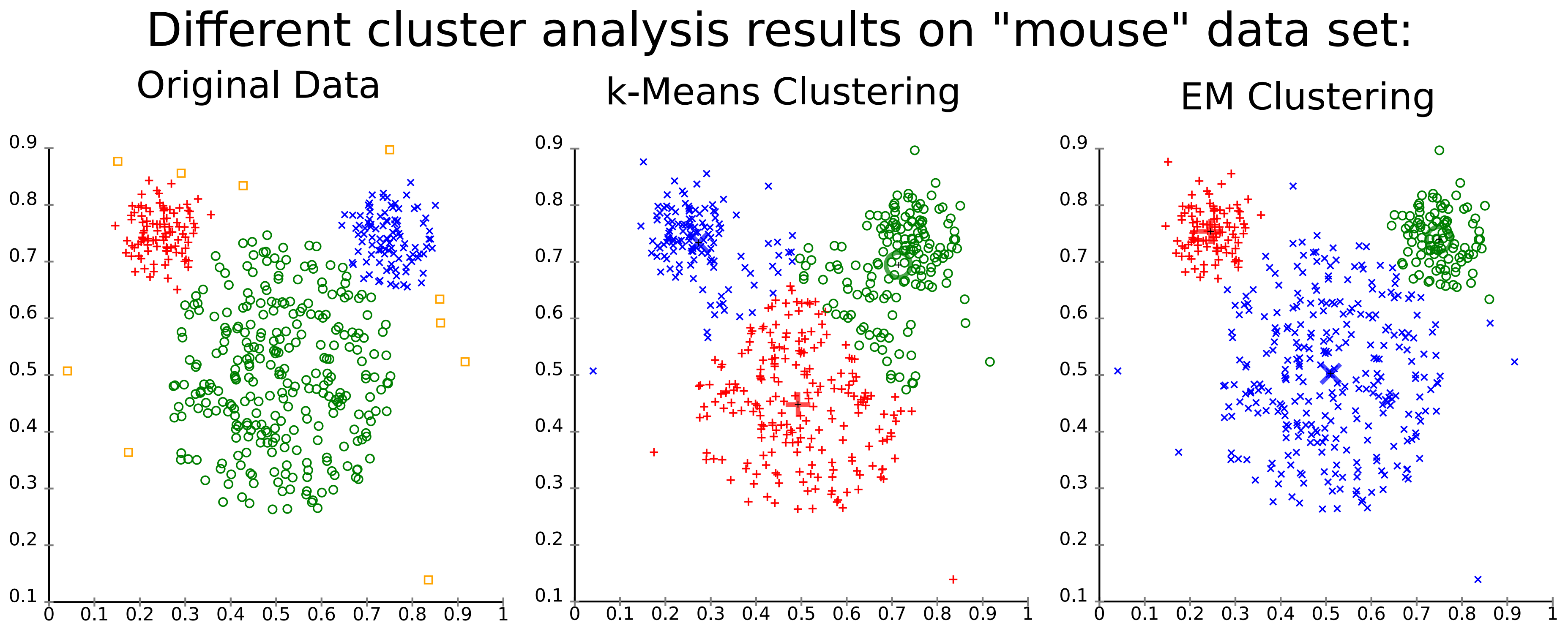


Figure 9.3 of Pattern Recognition and Machine Learning.

Gaussian Mixture vs K-Means

Clusters are modelled as Gaussians.

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

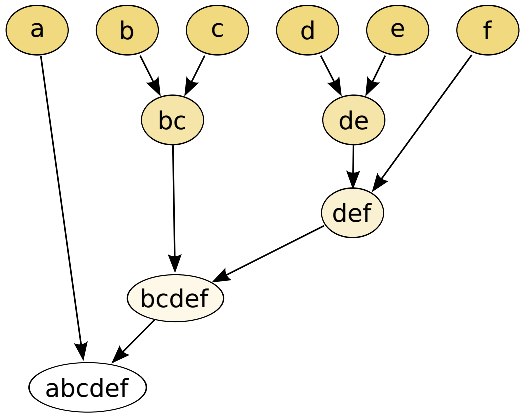


https://commons.wikimedia.org/wiki/File:ClusterAnalysis_Mouse.svg

Other types of clustering

Hierarchical clustering

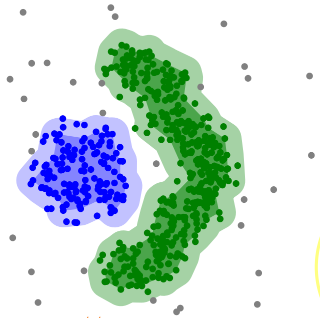
- Initially, each point is a singleton cluster
- Merging the most similar clusters until we reach the desired number of clusters
- Dozens of used metrics



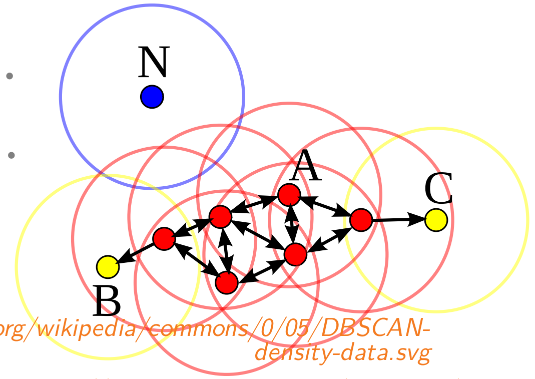
https://upload.wikimedia.org/wikipedia/commons/a/ad

Density based clustering

- Similar to hierarchical clustering
- Finds dense regions where at least `min_points` are in an ϵ diameter, connects everything within ϵ distance

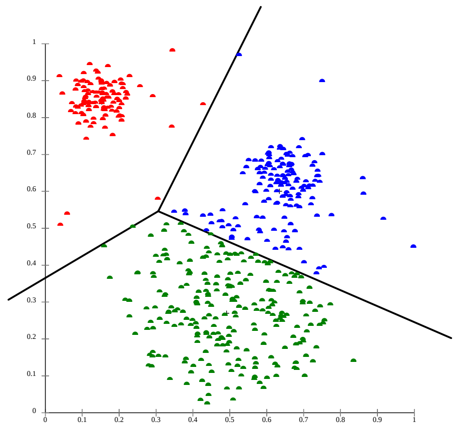
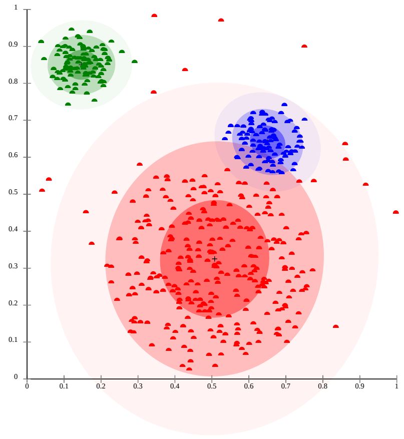
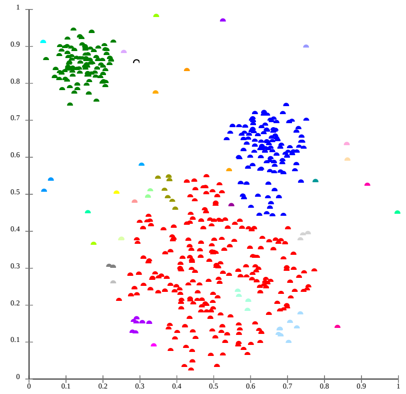
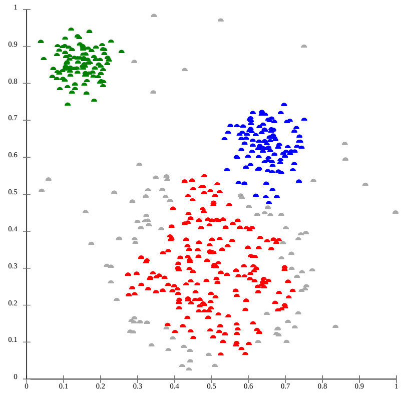


https://upload.wikimedia.org/wikipedia/commons/0/05/DBSCAN-density-data.svg

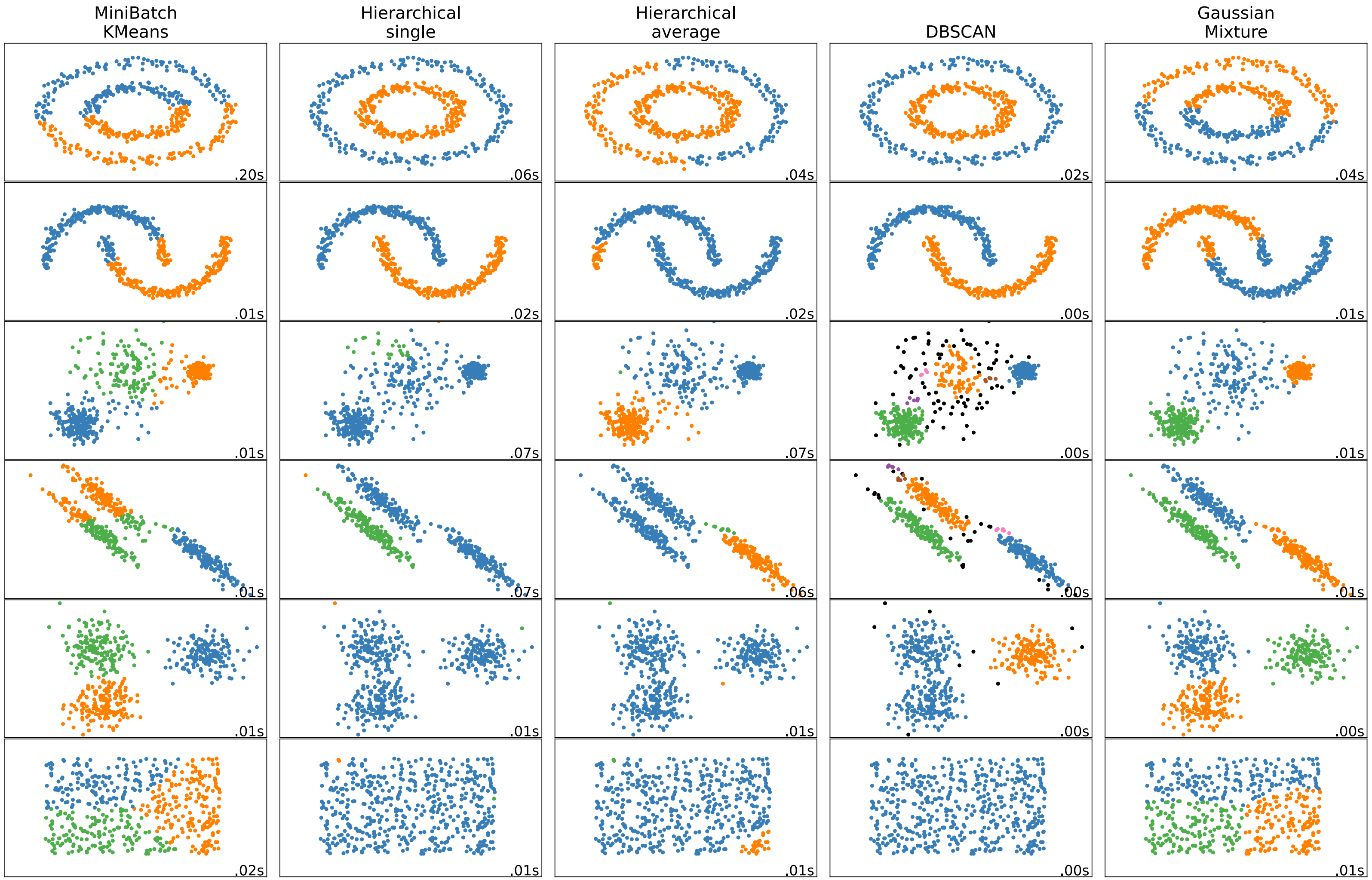


https://upload.wikimedia.org/wikipedia/commons/a/af/
Illus

Clustering algorithms compared (1)

K Means	Gaussian Mixture	Hierarchical	DBSCAN
 https://upload.wikimedia.org/wikipedia/commons/e/e5/KMeans-Gaussian-data.svg	 https://upload.wikimedia.org/wikipedia/commons/d/d8/EM-Gaussian-data.svg	 https://upload.wikimedia.org/wikipedia/commons/b/b7/SLINK-Gaussian-data.svg	 https://upload.wikimedia.org/wikipedia/commons/2/28/ Gaussia

Clustering algorithms compared (2)



https://scikit-learn.org/1.5/auto_examples/cluster/plot_cluster_comparison.html

Today's Lecture Objectives

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
- Implement the k-means algorithm and use it for clustering