Unsupervised learning

Eduardo Hugo Sanchez

December 10, 2021

Outline

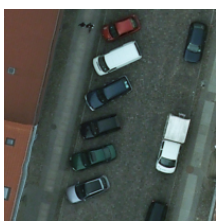
Supervised learning

- Learn a mapping from the input domain $\mathcal{X}$ to the output domain $\mathcal{Y}$, i.e. $f : \mathcal{X} \to \mathcal{Y}$.
- A dataset $X = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ consisting of inputs $x^{(i)}$ and their corresponding expected outputs $y^{(i)}$ is available
- The mapping f could represent many tasks:

Vegetation

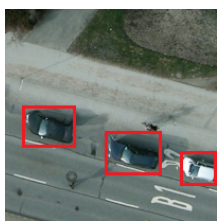
Input x Output y

Image classification



Input x Output y

Image segmentation

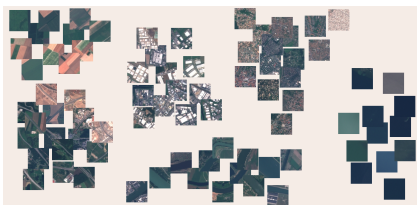


Input x Output y

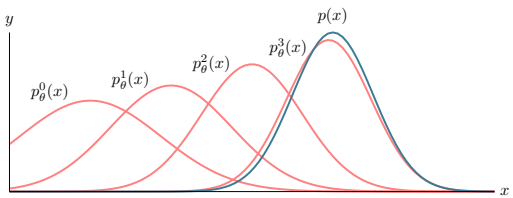
Object detection

Unsupervised learning

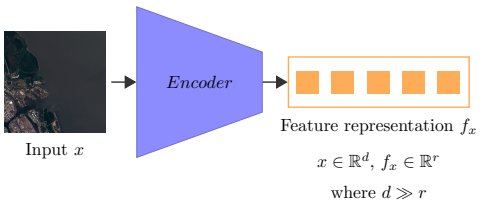
- There are no labels $y^{(i)}$ available associated to inputs $x^{(i)}$.
- Easier to create since we only need to collect data samples $x^{(i)}$ without providing labels, i.e. $X = \{x^{(i)}\}_{i=1}^{N}$.
- Learning the underlying hidden structure of data, i.e. the data distribution $p(x)$ *somehow*.



Clustering



Density estimation



Dimensionality reduction

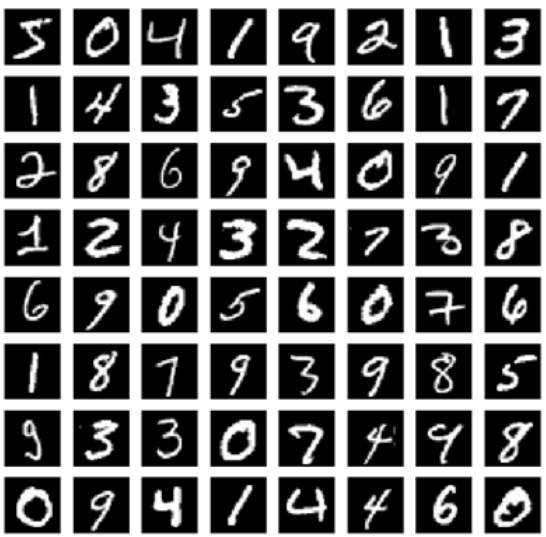
Supervised vs unsupervised learning

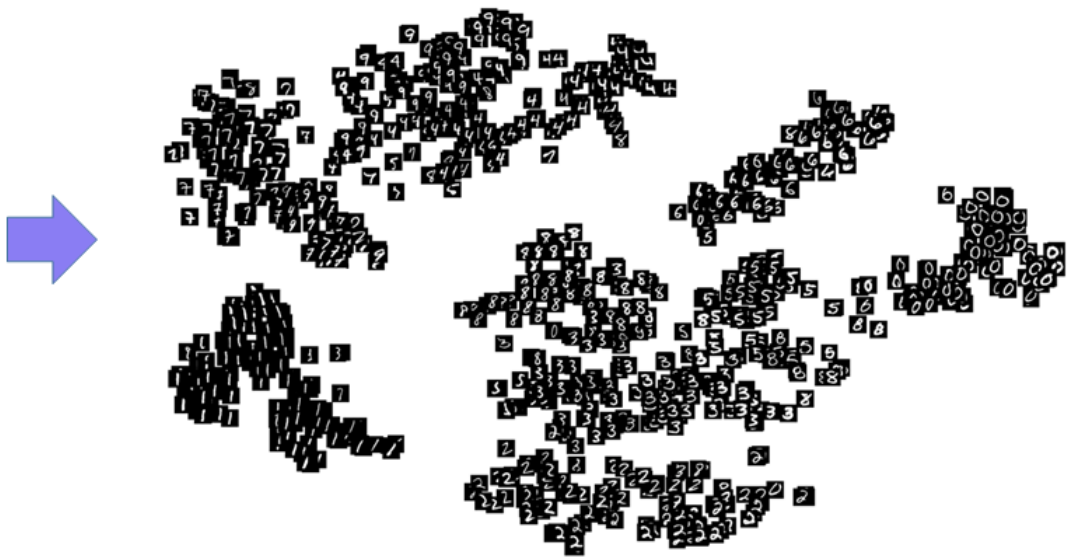
- Supervised learning focuses on learning the conditional probability distribution $p(y \mid x)$ while unsupervised learning learns the data distribution $p(x)$.
- To solve a given task, supervised learning learns a suitable data representation from raw data.
- Even though unsupervised learning does not solve a specific task (e.g. classification), it creates an internal data representation to learn $p(x)$.

Clustering

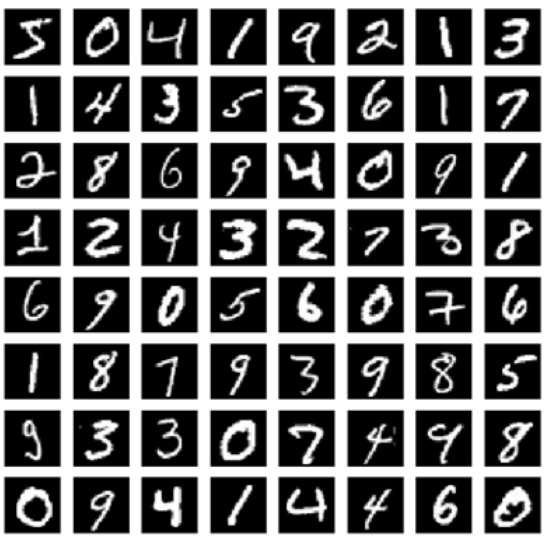
- Clustering is a machine learning method that aims to group a set of samples under certain criteria.
- Samples belonging to the same cluster are more similar (e.g. same class) to each other than to samples in other clusters.
- No labels are provided.

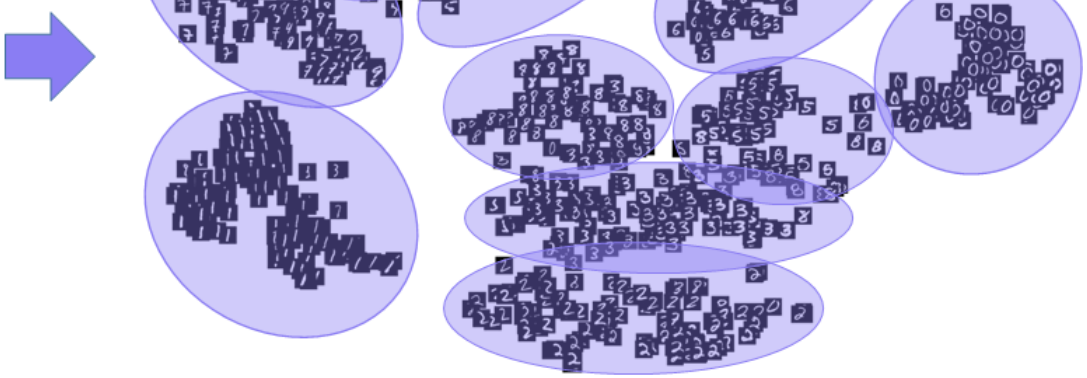
MNIST example



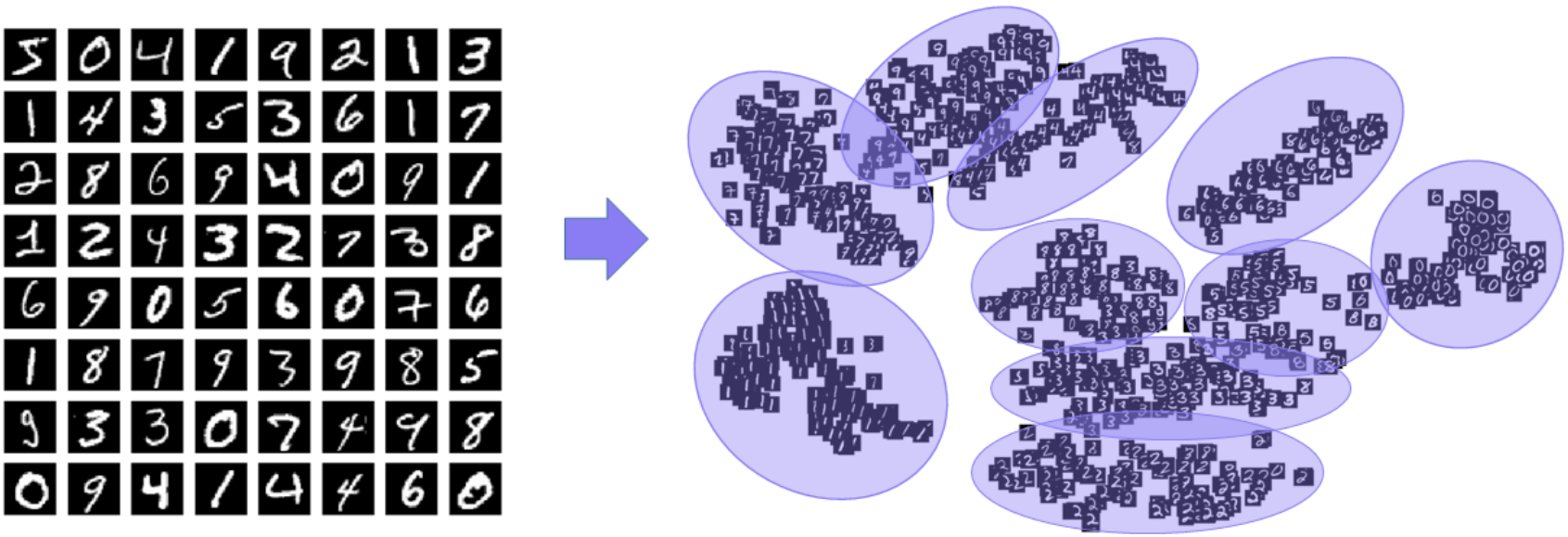


MNIST example





MNIST example



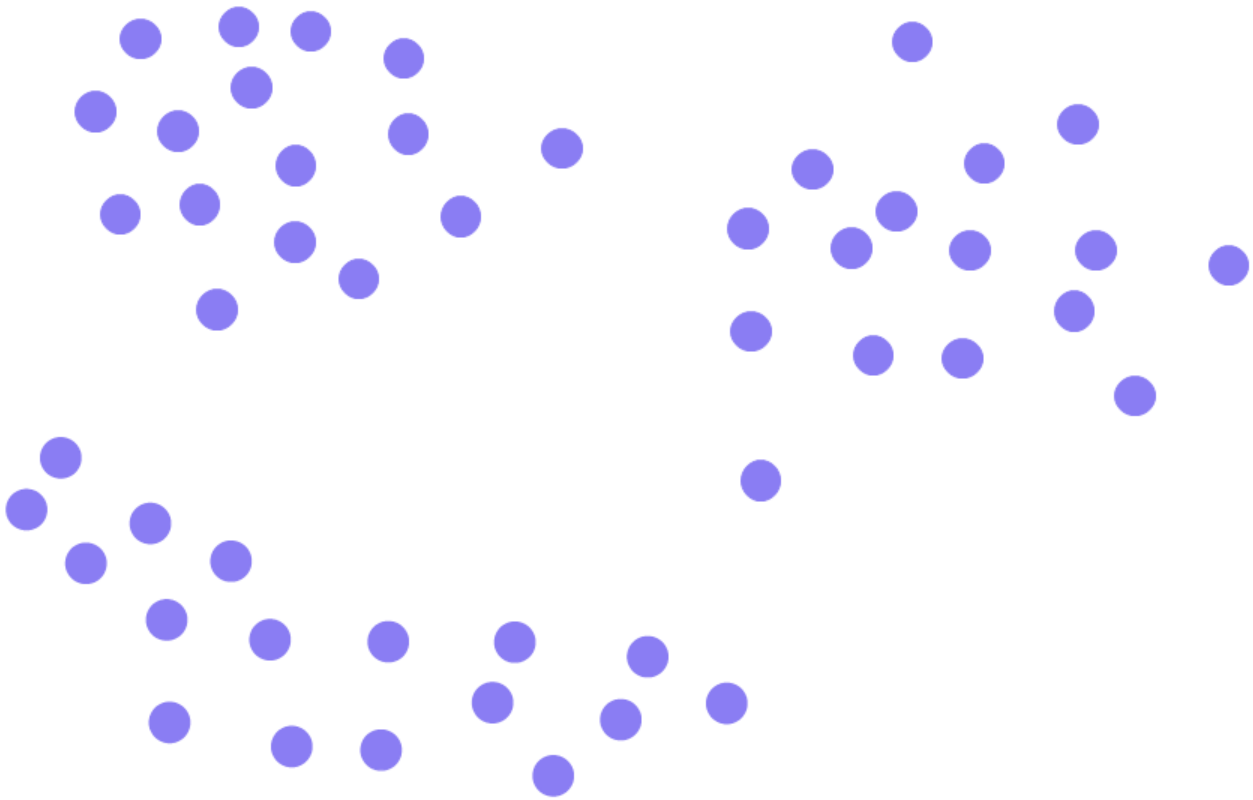
How to perform clustering on the MNIST dataset?

K-means

- It's a clustering algorithm that aims to partition n observations into k clusters.
- The following steps are performed:
  1. Initialize centroids
  2. Compute distances
  3. Assign clusters
  4. Update centroids
  5. Iterate from 2) until convergence

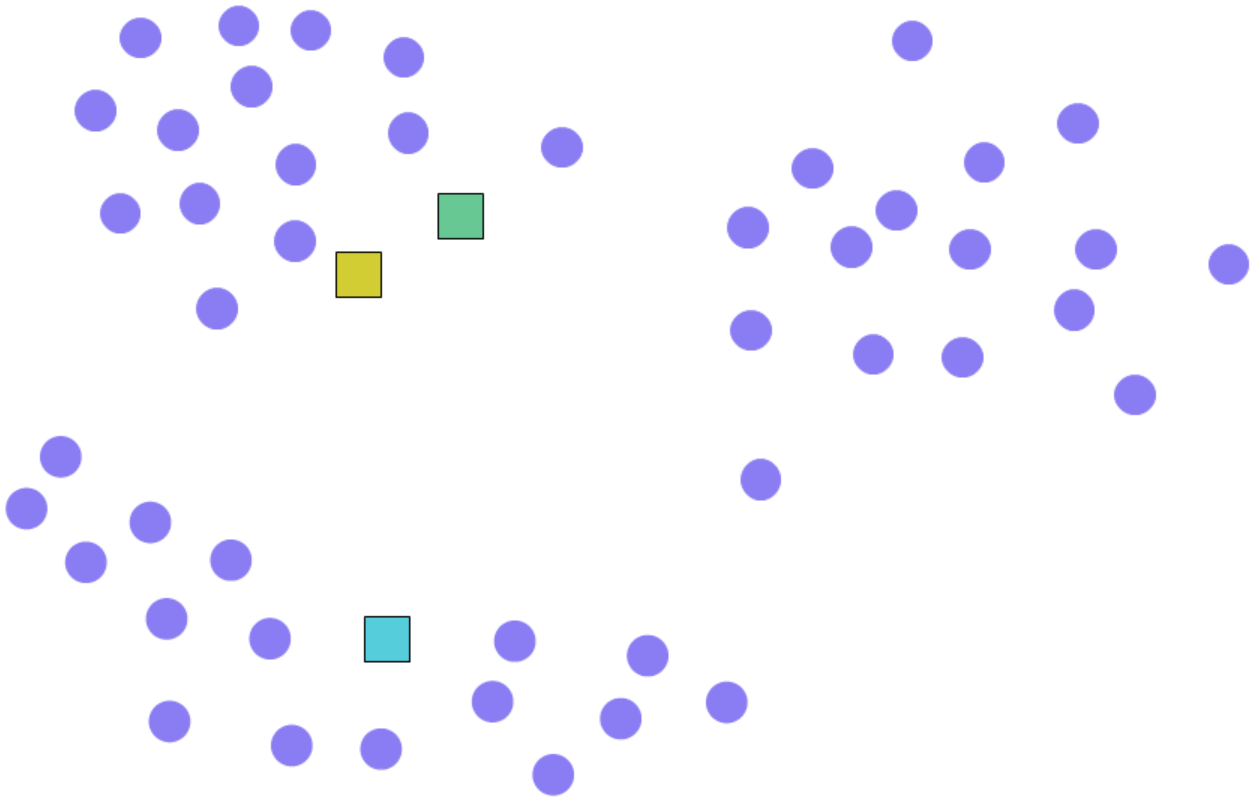
K-means: A 2D example

No labels are provided, k-means is performed on raw data.



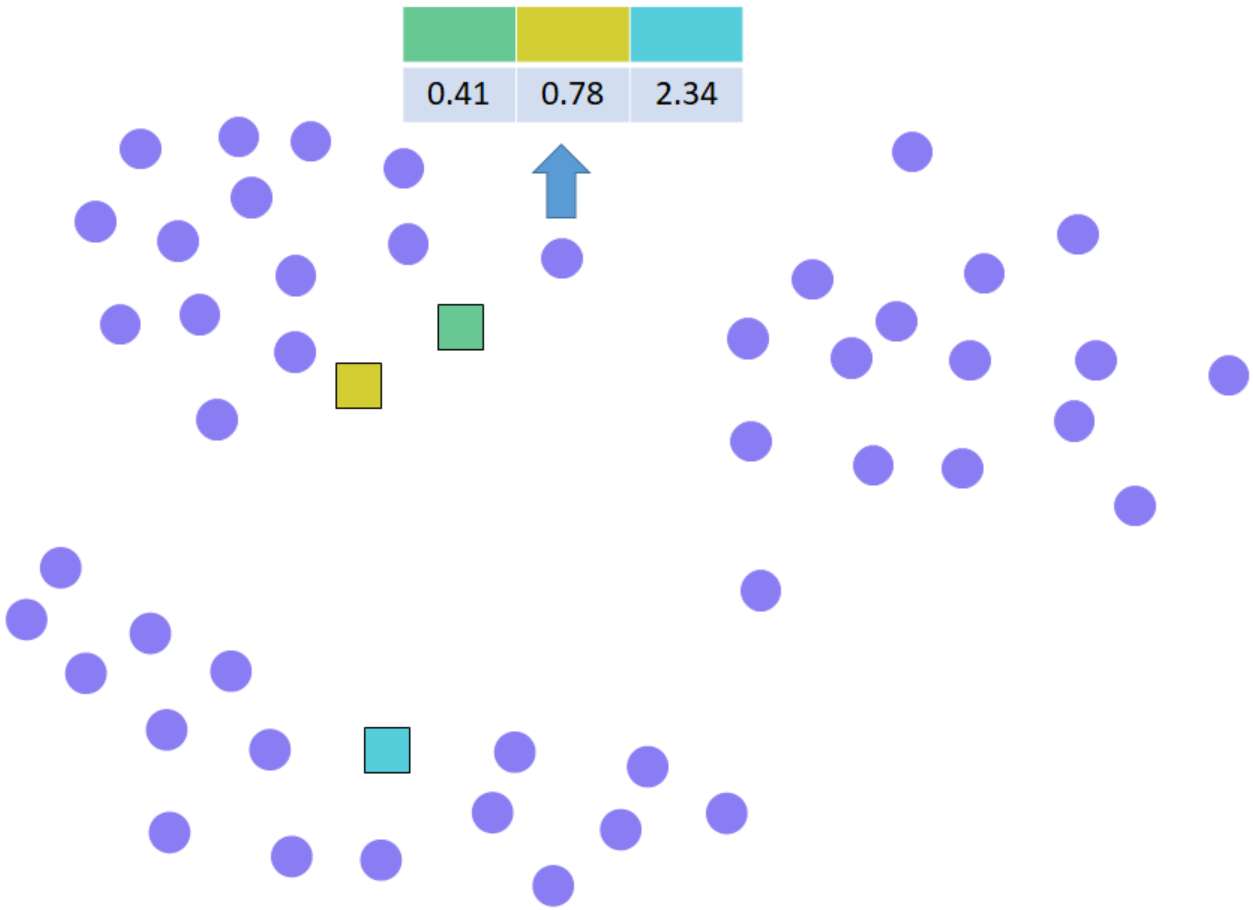
K-means: Initializing centroids

We define the number of clusters and initialize the centroids.



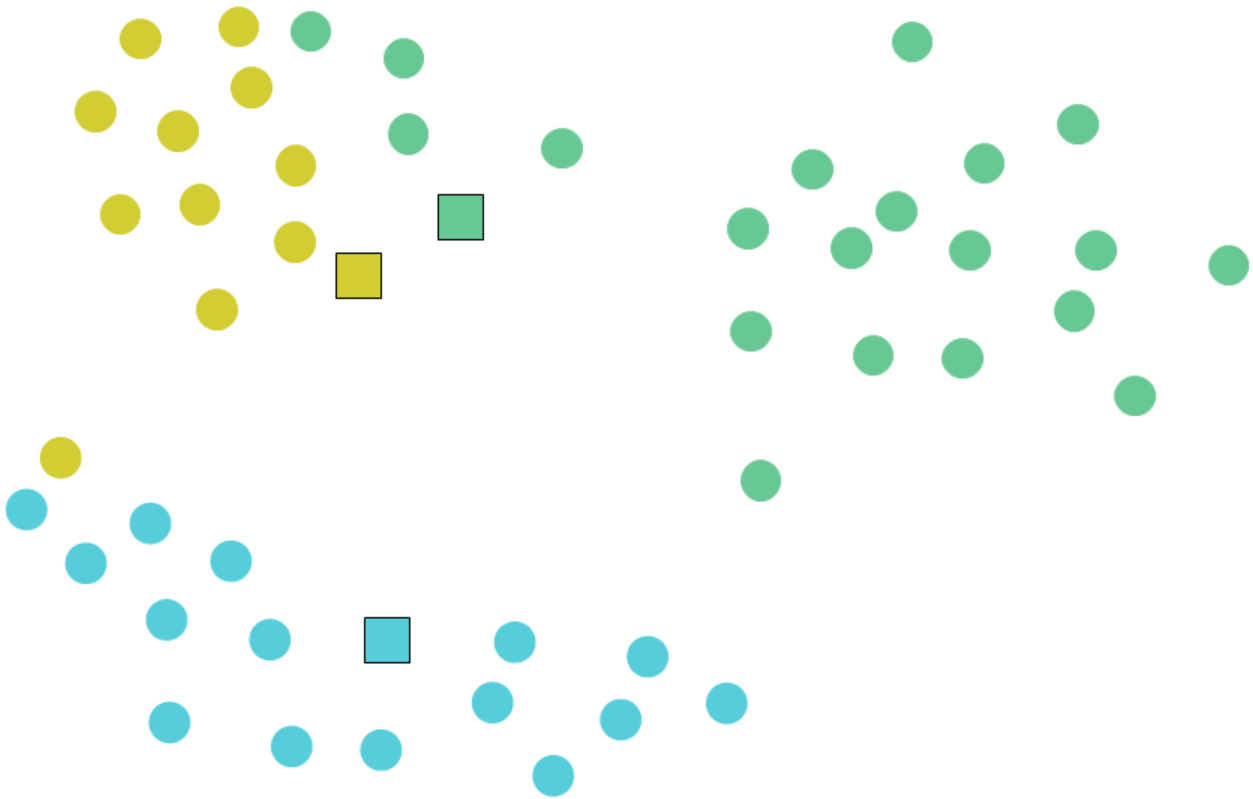
K-means: Computing distances

We compute distances between centroids and data samples.



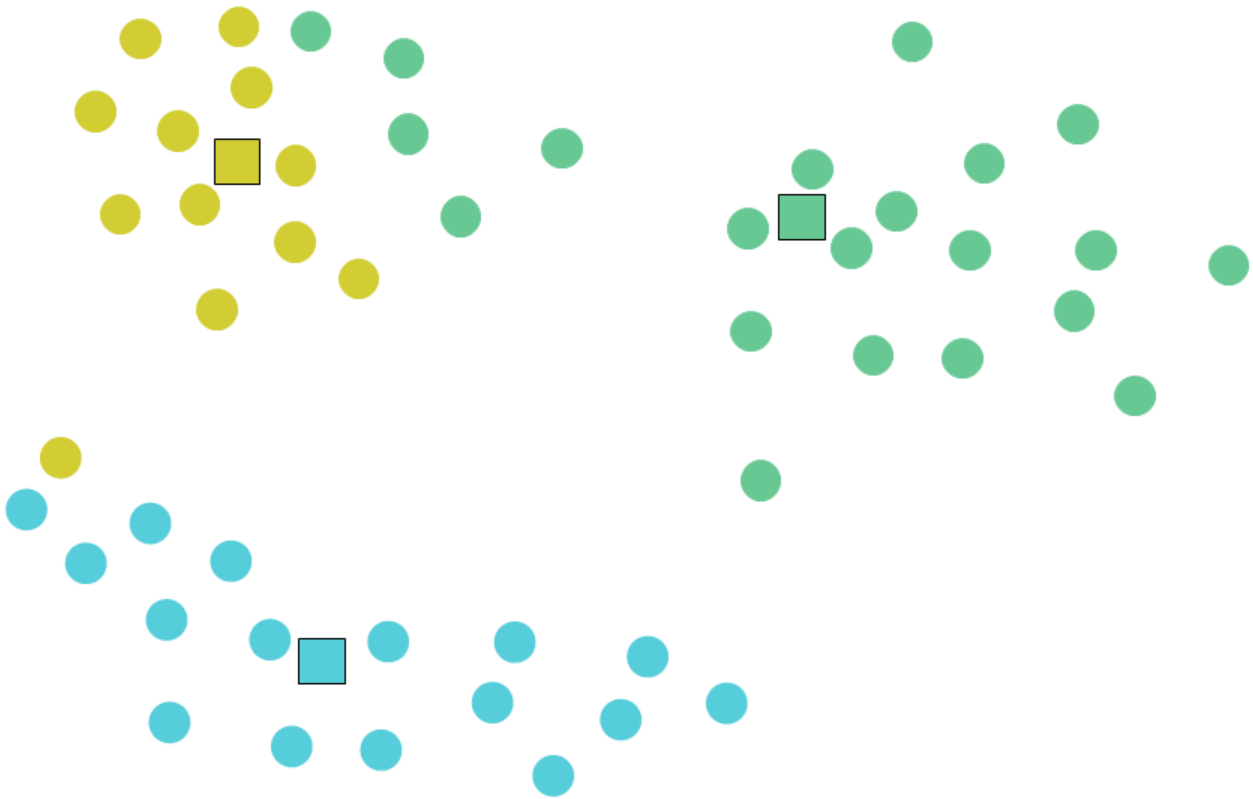
K-means: Assigning clusters

Based on the distances, we assign each data sample to a cluster.



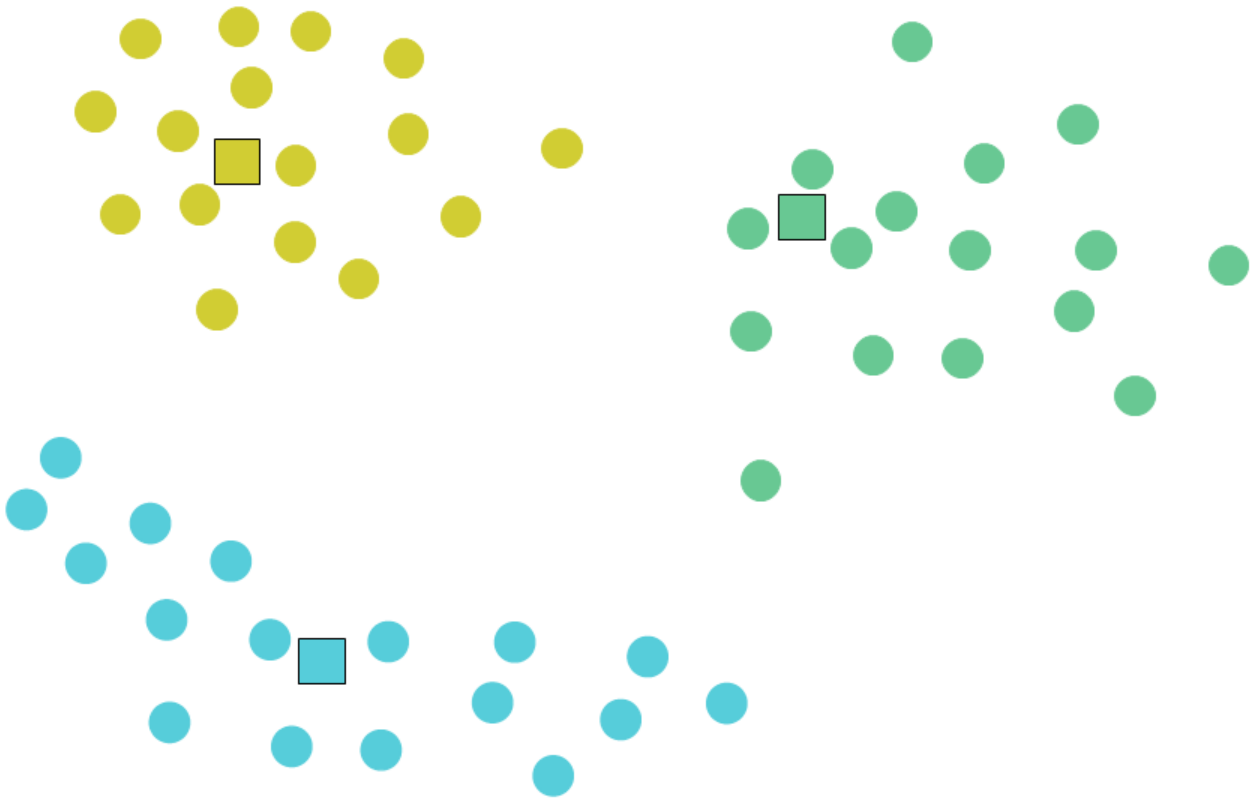
K-means: Updating centroids

Centroids are updated by averaging the samples in each cluster.



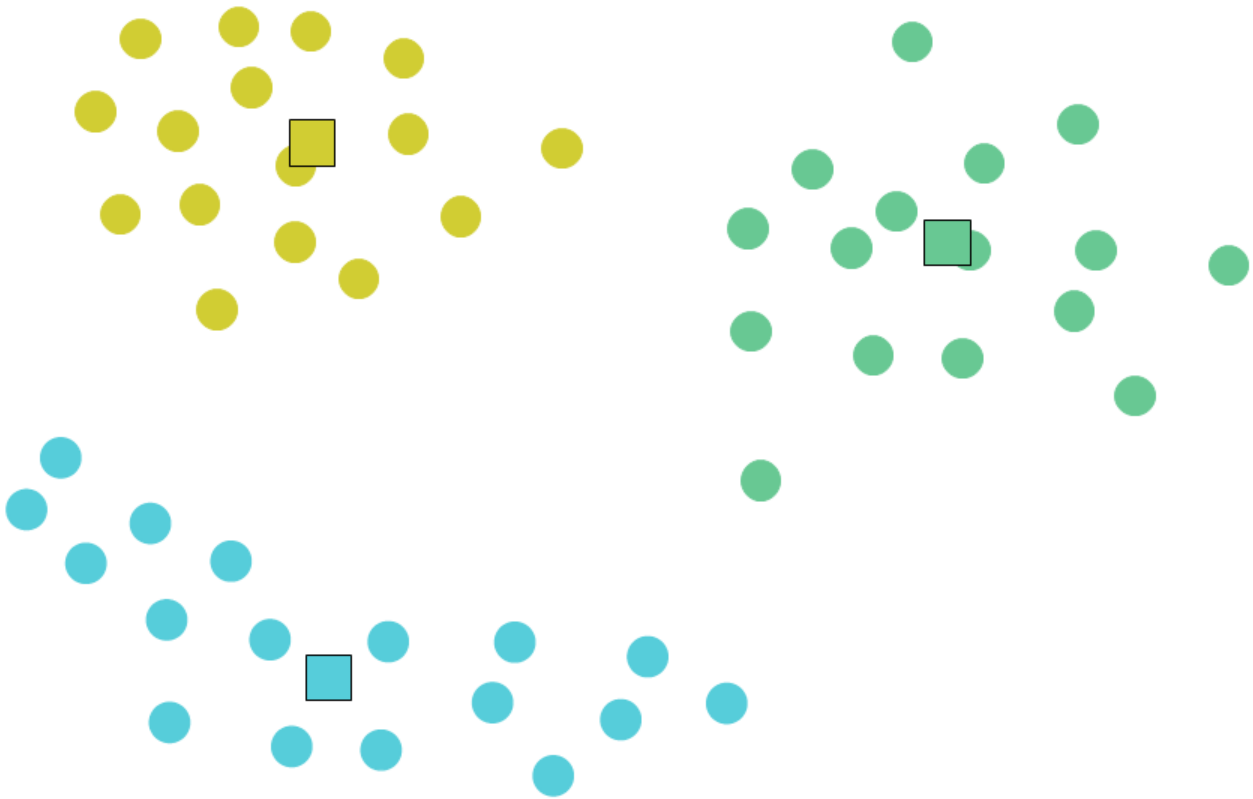
K-means: Assigning clusters

With the new centroids, we assign each data sample to a cluster.



K-means: Updating centroids

With the new clusters, we updated the centroids.



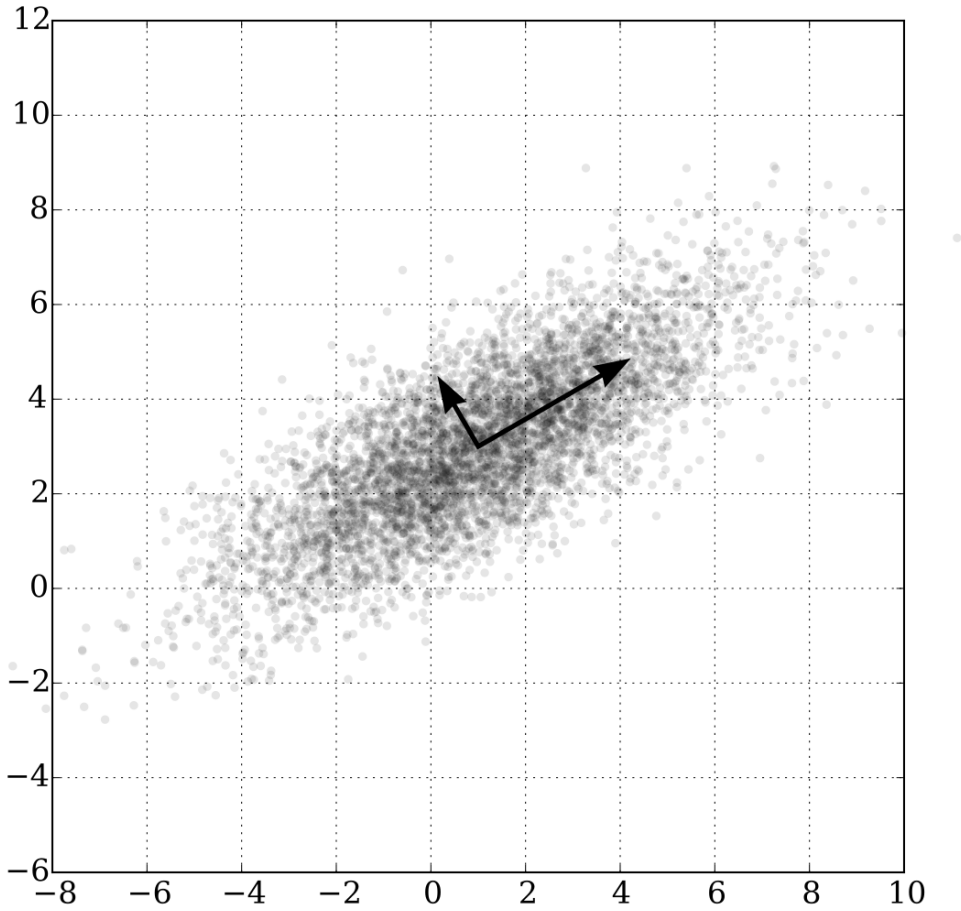
Dimensionality reduction

- A data sample is typically defined by a high-dimensional variable (e.g. a 1024x1024 pixel image = 1048576 dimensions)
- A large number of dimensions can result in poor performance at some tasks.
- Dimensionality reduction techniques transform data from a high-dimensional space into a low-dimensional space while keeping the relevant information from data.

Principal component analysis (PCA)

- PCA consists in computing an orthonormal basis and using it to perform a change of basis on the data.
- The first principal component corresponds to the direction that maximizes the variance of the projected data. The next principal component corresponds to the direction orthogonal to the previous principal component that maximizes the variance of the projected data.
- Dimensionality reduction is carried out by keeping the first components of the orthonormal basis and ignoring the remaining components.

A simple 2D example



Principal component analysis (PCA)

Given a set of N samples x_1, ..., x_N, we create the matrix of the dataset as follows

$$X = \begin{pmatrix} — & x_1 & — \\ — & x_2 & — \\ — & x_3 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

Principal component analysis (PCA)

We subtract the mean from the dataset.

$$\bar{X} = \begin{pmatrix} — & x_1 & — \\ — & x_2 & — \\ — & x_3 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix} - \mu_X = \begin{pmatrix} — & (x_1 - \mu_X) & — \\ — & (x_2 - \mu_X) & — \\ — & (x_3 - \mu_X) & — \\ & \vdots & \\ — & (x_N - \mu_X) & — \end{pmatrix}$$

Where μ_X is the mean of X.

Principal component analysis (PCA)

We compute the covariance matrix $C = \bar{X}.\bar{X}^T$.

$$\bar{X}.\bar{X}^T = \begin{pmatrix} — & (x_1 - \mu_X) & — \\ — & (x_2 - \mu_X) & — \\ & \vdots & \\ — & (x_N - \mu_X) & — \end{pmatrix} . \begin{pmatrix} | & | & & | \\ (x_1 - \mu_X) & (x_2 - \mu_X) & \dots & (x_N - \mu_X) \\ | & | & & | \end{pmatrix}$$

Principal component analysis (PCA)

We compute the eigenvectors W of the covariance matrix C.

$$C = W.\Sigma.W^T$$

(see the the function numpy.linalg.eig() to compute the matrix W)

Principal component analysis (PCA)

We compute the principal components T of our data as follows.

$$T = X.W$$

t-distributed stochastic neighbor embedding (t-SNE)

- t-SNE is a statistical method for visualizing high-dimensional data by mapping each data sample to 2D or 3D vectors.
- Three main stages are carried out:
  1. Computing similarity between data points in the high-dimensional space.
  2. Transforming data points into 2D or 3D vector and computing similarity between these low-dimensional representations.
  3. Forcing the similarity between data points in the high-dimensional space to be close to the similarity in the low-dimensional space.

t-distributed stochastic neighbor embedding (t-SNE)

Given a set of N samples x_1, ..., x_N, t-SNE first computes probabilities $p_{i,j}$ that are proportional to the similarity between the samples x_i and x_j:

$$p_{i,j} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{i \neq k} \exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

Where $p_{i,i} = 0$.

t-distributed stochastic neighbor embedding (t-SNE)

t-SNE learns low-dimensional vectors y_1, ..., y_N from the samples x_1, ..., x_N. Similarity between the vectors y_i and y_j is computed as follows:

$$q_{i,j} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}}$$

Where $q_{i,i} = 0$.

t-distributed stochastic neighbor embedding (t-SNE)

Vectors y_i are learned by minimizing the Kullback–Leibler divergence between the distribution P and the distribution Q. This is done via gradient descent.

$$loss = D_{KL}(P\|Q) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)})$$

t-distributed stochastic neighbor embedding (t-SNE)

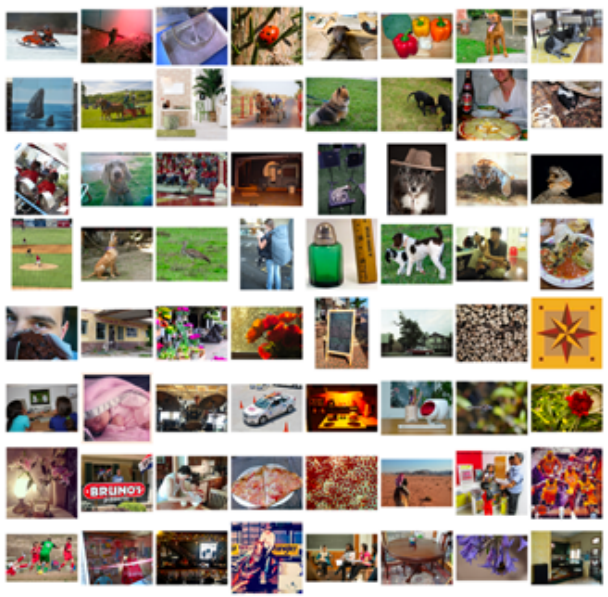
To update y_i, the derivative of the loss is required:

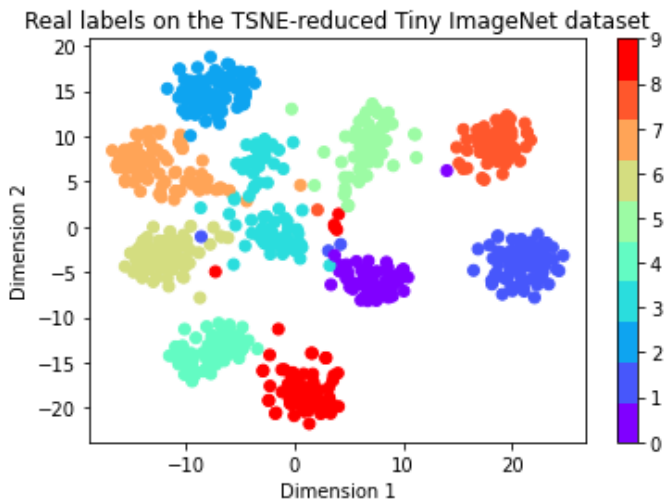
$$\frac{\partial loss}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$$

Deep-learning-based methods

- We learn from data the way dimensionality reduction is carried out (e.g. autoencoders, generative models, etc.).
- We define the properties we keep in the low-dimensional representation.

ImageNet example





Q&A

Thank you for your attention!