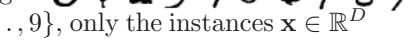
CSE176 Introduction to Machine Learning
Fall semester 2015

Lab: clustering algorithms
Miguel Á. Carreira-Perpiñán

The objective of this lab is for you to program several representative clustering algorithms in Matlab, apply them to some datasets and observe their behavior. The TA will first demonstrate the results of the algorithms on a toy dataset and the MNIST dataset, and then you will program them, replicate those results, and further explore the datasets with the algorithms. You can use the textbook, lecture notes and your own notes.

I Datasets

Construct your own toy datasets in 2D, such as Gaussian clusters with more or less overlap, or clusters with curved shapes as in the 2moons dataset. You will also use the MNIST dataset of handwritten digits  . Since clustering algorithms are unsupervised, they do not use the class labels $y_n \in \{0, \dots, 9\}$, only the instances $\mathbf{x} \in \mathbb{R}^D$ (where $D = 784$). You can use the labels to see if they agree with the resulting clusters found by an algorithm.

II Implementing and using clustering algorithms

Implement the following algorithms: k-means, EM for Gaussian mixtures with full covariances, mean-shift and connected-components. The figure shows pseudocode for the algorithms. For algorithms that take an infinite number of iterations to converge, stop them after `maxit` iterations (e.g. 100) or once the error function changes by less than a small value `tol` (e.g. 10^{-3}). With toy datasets in 2D, plot the following figures:

- For k-means:
  1. The value of the error function after each iteration. It should decrease monotonically and stop in a finite number of iterations.
  2. The dataset in 2D, with points colored according to the cluster they belong to.
- For EM with Gaussian mixtures:
  1. The value of the log-likelihood function after each iteration. It should increase monotonically.
  2. The dataset in 2D, with points colored according to the cluster they belong to. You can assign point $\mathbf{x}_n$ to cluster k if $p(k|\mathbf{x}_n) > p(j|\mathbf{x}_n)\ \forall j \neq k$.
  3. Even better, plot $p(k|\mathbf{x})$ itself for each class for $\mathbf{x} \in \mathbb{R}^2$ (as a color plot, or as a contour plot for each cluster).
- For mean-shift:
  1. The value of the density $p(\mathbf{x})$ after each iteration (initialized from each point $\mathbf{x}_n$). It should increase monotonically.
  2. The dataset in 2D, with points colored according to the cluster they belong to.
- For connected-components:
  1. The dataset in 2D, with points connected by edges in the ϵ-ball graph.
  2. The dataset in 2D, with points colored according to the cluster they belong to.

Explore each algorithm in different settings (for the same dataset):

- For algorithms that depend on the initialization (k-means and EM), try different random initializations.
- Try different values of the user parameter (number of clusters K for k-means and Gaussian mixtures with EM, bandwidth $\sigma > 0$ for mean-shift, scale $\epsilon > 0$ for connected-components).

With the MNIST dataset, try EM with Gaussian mixtures (also k-means) with different K values and plot:

1. The mean $\boldsymbol{\mu}_k$ of each cluster $k = 1, \dots, K$, as a grayscale image, with its mixing proportion $\pi_k = p(k)$.
2. The posterior probabilities $p(k|\mathbf{x}_n)$ for $k = 1, \dots, K$ for a given digit image $\mathbf{x}_n$, plotted as a bar chart.

The following Matlab functions will be useful (among others): `mean cov find randn linspace scatter contour gplot bar`.

k-means algorithm

$\{\boldsymbol{\mu}_k\}_{k=1}^K \leftarrow K$ random points from $\{\mathbf{x}_n\}_{n=1}^N$
repeat
  for $n \in \{1, \dots, N\}$
    $k^* = \arg\max_{k=1,\dots,K} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|$ — closest mean to $\mathbf{x}_n$
    $z_{nk^*} = 1$ and $z_{nk} = 0 \; \forall k \neq k^*$
  for $k \in \{1, \dots, K\}$
    $\boldsymbol{\mu}_k \leftarrow \sum_{n=1}^N z_{nk}\boldsymbol{\mu}_k / \sum_{n=1}^N z_{nk}$ — mean of points in cluster k
until stop
return $\{\boldsymbol{\mu}_k\}_{k=1}^K$, $\mathbf{Z}$

Gaussian mixture estimated with EM algorithm

Initialize $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ from k-means
repeat
  for $n \in \{1, \dots, N\}$
    $z_{nk} \leftarrow p(k|\mathbf{x}_n) =$ eq. (7.14) — E step
  for $k \in \{1, \dots, K\}$
    $\pi_k \leftarrow$ eq. (7.11), $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \leftarrow$ eq. (7.13) — M step
until stop
return $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$

Gaussian mean-shift algorithm

for $n \in \{1, \dots, N\}$
  $\mathbf{x} \leftarrow \mathbf{x}_n$
  repeat
    $\forall n$: $p(n|\mathbf{x}) \leftarrow \dfrac{\exp\left(-\frac{1}{2}\|(\mathbf{x}-\mathbf{x}_n)/\sigma\|^2\right)}{\sum_{n'=1}^N \exp\left(-\frac{1}{2}\|(\mathbf{x}-\mathbf{x}_{n'})/\sigma\|^2\right)}$
    $\mathbf{x} \leftarrow \sum_{n=1}^N p(n|\mathbf{x})\mathbf{x}_n$
  until stop
  $\mathbf{z}_n \leftarrow \mathbf{x}$ — mode found from $\mathbf{x}_n$
end
return connected-components$(\{\mathbf{z}_n\}_{n=1}^N, \epsilon)$ — aggregate modes found

Connected-components algorithm

Define an ϵ-ball graph:

- vertices $\mathbf{x}_1, \dots, \mathbf{x}_N$
- edges $(\mathbf{x}_n, \mathbf{x}_m) \Leftrightarrow d(\mathbf{x}_n, \mathbf{x}_m) < \epsilon$, $\forall n, m = 1, \dots, N$.

Apply DFS to this graph.
return its connected components

Figure 1: Pseudocode for k-means, EM for Gaussian mixtures, mean-shift for the Gaussian kernel and connected-components for an ϵ-ball graph (with a distance function $d(\cdot, \cdot)$ applicable to any pair of points). In all cases, the input is a dataset $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^D$ and a user parameter: number of clusters K for k-means and Gaussian mixtures with EM, bandwidth $\sigma > 0$ for mean-shift, and scale $\epsilon > 0$ for connected-components.

Practical advice:

- Many operations in machine learning algorithms involve vectors and matrices. Try to program these using vector and matrix operations in Matlab ("vectorized code") rather than loops, because 1) the code will be shorter and more readable (closer to the pseudocode), 2) it will be faster (because Matlab is an interpreted language), and 3) you will save effort and avoid bugs. Example: a matrix-vector product `y = A*x` instead of a double loop $y_i = \sum_{j=1}^n a_{ij}x_j$ for $i = 1, \dots, n$.
- Machine learning algorithms can have a high time or space complexity, so to get a result in a few seconds you may need to run them on small datasets. You can do this by selecting a random sample of a given dataset.
- Machine learning algorithms often are randomized. Likewise, toy datasets are usually generated randomly. To make sure you can generate the exact dataset multiple times and run an algorithm and get the same result every time, fix the seed of the pseudorandom number generator. In Matlab: `rng(1778);` where 1778 is the seed. You can also save a toy dataset for later use.
- Matlab tips:
  - To suppress extra line feeds: `format compact`.
  - To get more decimals: `format long`.
  - To compare two matrices or vectors (by finding the largest difference): `max(abs(A(:)-B(:)))`.
  - To avoid distorted plots: `daspect([1 1 1])`.
  - To plot grayscale images with values in $[0, 1]$: `colormap(gray(256)); imagesc(I,[0 1]);`
    To plot images with negative and positive values: `colormap(parula(256)); imagesc(I);`