The objective of this lab is to give you with a practical tour of data analysis and machine learning (ML) from the point of view of a user. Specifically, given a dataset of multidimensional vectors, you will discover some of the structure in the data by means of basic visualization techniques and machine learning algorithms, and explore different parameter settings for the latter. At this stage, you will not understand how a given algorithm works, but you should understand what problem it solves and how to use it in practice—by giving appropriate input data to the algorithm, setting its hyperparameters, running it, and interpreting and using its output. You will do this by downloading several datasets and code from the web. All the code will be in Matlab; get familiar with Matlab programming if you are not already.

I Datasets

You will use a representative selection of "toy" and "real-world" datasets. Toy datasets consist of low-dimensional vectors (1D to 3D) that can be easily visualized and are typically synthetically generated (e.g. a set of Gaussian clusters or a noisy line). They are useful to understand the behavior of an algorithm in a tightly controlled experiment. Real-world datasets have been obtained from some actual problem (e.g. photos or speech) and usually have both a large dimensionality and a large sample size.

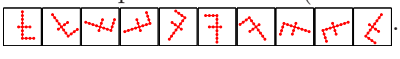
In unsupervised learning, a dataset consists of a set of N vectors in D dimensions $\{\mathbf{x}_n\}_{n=1}^N$, which we store in Matlab as a matrix of $N \times D$ or $D \times N$. In supervised learning, we have N pairs of vectors $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ where $\dim(\mathbf{x})$ may be different from $\dim(\mathbf{y})$.

Toy datasets We usually generate them with some known function or probability distribution. Examples:

- *Noisy function* $y = f(x) + \epsilon$ from 1D to 1D. We can vary the function f and the level and distribution of the noise ϵ. This is useful for (nonlinear) regression.
- *Gaussian clusters* in 2D. We can vary the number K of clusters, their location (mean $\boldsymbol{\mu}_k$), their shape (covariance matrix $\boldsymbol{\Sigma}_k$) and their proportion of points (π_k). Use `GMsample` from the GM tools (or use `randn` appropriately). This is useful for clustering, classification and other things.
- *Spirals* in 2D. This is useful for clustering, classification and other things.
- *Two moons* dataset in 2D. This is useful for clustering, classification and semisupervised learning.
- *Swiss roll* dataset in 3D. This is useful for dimensionality reduction (manifold learning).

Be creative. Generate your own toy datasets.

Real-world datasets

- *MNIST* (`http://yann.lecun.com/exdb/mnist`): grayscale images of 28×28 pixels of handwritten digits, such as  . Each image correspond to one instance $\mathbf{x}_n$ and is represented as a vector with $28 \times 28 = 784$ dimensions.
- *COIL* (`http://www.cs.columbia.edu/CAVE`): grayscale images of 128×128 pixels of 20 objects rotated every 5°, such as  .
- *Wisconsin X-ray microbeam data (XRMB)*
- Tongue shapes
- MNIST rotated digit-7 and digit skeleton dataset (`http://faculty.ucmerced.edu/mcarreira-perpinan/software.html`). We created this by selecting 40 digits '7' at random from the MNIST dataset, adding some noise and rotating each digit by 6-degree increments from 0 to 360 degrees, for a total of $N = 2\,400$ images of 28×28:  . Also, for each digit-7 image, we manually created a point-set in 2D (with 14 points, hence a vector of dimension 28) representing the skeleton of the digit:  . This dataset can be used to learn to map digit-7 images to skeletons.

Many datasets of manageable size, widely used as benchmarks in ML, are in the UCI repository (`http://www.ics.uci.edu/~mlearn/MLRepository.html`).

II Visualizing high-dimensional data

We can plot the entire dataset $\{\mathbf{x}_n\}_{n=1}^N$ as a point cloud only when the data has dimension $D \leq 3$. To visualize data with higher dimension, these are some basic techniques:

- Scatterplots of pairs of variables: plot the component i vs the component j of each data vector $\{\mathbf{x}_n\}_{n=1}^N$.
- Histogram of a single variable i, or 2D histogram of a pair of variables i and j.
- Plot an *individual* vector $\mathbf{x}_n$ (for a given n) depending on its meaning:
  - If $\mathbf{x}_n$ is an image, reshape it and plot it as an image (e.g. for MNIST $784 \to 28 \times 28$).
  - If $\mathbf{x}_n$ is a curve consisting of $D/2$ points in 2D, reshape it and plot it as a curve in 2D (e.g. tongue shapes dataset).
  - If $\mathbf{x}_n$ is a time series, plot it as a time series.

  And then we can loop over plotting instances $n = 1, 2, \ldots$, or combine them into a single figure.

You can apply some of these techniques to plotting the parameters estimated by a ML algorithm, such as the component means in a Gaussian mixture.

It is also informative to examine basic statistics of the data, such as its mean; covariance matrix; maximum, minimum and range over each dimension; etc.

III Machine learning algorithms

Below is a set of ML algorithms, corresponding to different ML problems (clustering, classification, etc.). Even though you probably will not understand how each algorithm works, you should understand what is expected from it, and explore its behavior. You can apply them to toy and real-world datasets. For each algorithm, explore different aspects of its performance:

- With toy datasets, as mentioned earlier, vary the dataset distribution (number and shape of the clusters, density) or generating function (linear, nonlinear); the sample size N; the dimensionality D; the intrinsic dimensionality; etc.
- Vary different parameters of the algorithm:
  - Each algorithm usually requires the user to specify some "hyperparameters" that help find a good solution. For example: for clustering, the desired number of clusters K or a scale parameter σ; for regression or classification, the number of parameters in the function (number of weights, basis functions, etc.); for distance- or similarity-based algorithms, the number of nearest neighbors k or a scale parameter σ; for kernel density estimation, a scale parameter σ; etc.
  - Many algorithms have a "regularization" parameter λ that controls the tradeoff between smoothness and goodness of fit.
  - In addition, there may be parameters that control the optimization: maximum number of iterations, error tolerance to stop training, etc.

How well does the algorithm work in all these situations? If it makes mistakes, can you understand why?

ML algorithms

- Clustering:
  - K-means.
  - Gaussian mixtures and EM algorithm.
  - Laplacian K-modes.
  - Image segmentation and mean-shift clustering (applet).
- Density estimation:
  - Kernel density estimation.

  - Gaussian mixtures.

- Classification:
  - Linear classification using support vector machines (SVMs).
  - Nonlinear classification using kernel SVMs and low-dimensional SVMs.
- Regression:
  - Linear regression.
  - Nonlinear regression using radial basis function (RBF) networks.
- Dimensionality reduction:
  - Linear dimensionality reduction: PCA.
  - Nonlinear dimensionality reduction: Laplacian eigenmaps; elastic embedding.
- Missing data reconstruction (pattern completion):
  - Gaussian mixtures.

Practical advice:

- 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 avoid distorted plots: `daspect([1 1 1])`.