Machine Learning An Algorithmic Perspective: Sections 6.4 ICA, 6.5 LLE, 6.6 MDS & Isomap

By Alex Milne

6.4 INDEPENDENT COMPONENTS ANALYSIS (ICA)

- Motivation for ICA is the problem of “blind” source separation
- For PCA, components chosen that were orthogonal and uncorrelated cov(bi, bj) = 0 if i != j
- ICA requires that the components are statistically independent (so that for E[bi , bj] = E[bi]E[bj] as well being uncorrelated)
- Assumption: data are actually created by a set of physical processes that are independent => data we see are correlated => different processes have been mixed together => find a transformation that turns it into a mixture of independent sources or components.
- Cocktail party.

ICA Algorithm

Suppose that we have two sources making noise (s1 , s2)

And two microphones that hear things, giving inputs (x1 , x2)

Need ear/ microphone for each source.

The sounds that are heard from the sources as:

$$\begin{aligned} x_1 &= as_1 + bs_2, \\ x_2 &= cs_1 + ds_2, \end{aligned}$$

=> Matrix form => $\mathbf{x} = \mathbf{A}\mathbf{s},$ A is known as the mixing matrix.

- Undo the mixing by doing the inverse of the mixing matrix (xA^-1) but we don't know A ☹
- What we know about the sources and the signals:
  1. The mixtures are not independent, even though the sources are
     ⇒ if we find factors that are independent of each other then they are probably sources
  2. The mixtures will look like normal distributions even if the sources are not (because of the Central Limit Theorem)
     ⇒ if we find factors that are not Gaussian then they are probably sources
  3. The mixtures will look more complicated than the sources

6.4 INDEPENDENT COMPONENTS ANALYSIS (ICA)

- We find independence between two variables by using the **mutual information (Section 12.2.1 entropy)**
- Most common approach => negentropy: J(y) = H(z) − H(y) maximises the deviations from Gaussianness (where H(·) is the entropy)

$$H(y) = -\int g(y) \log g(y) dy.$$

- Common approximation:

$$(E[G(y)] - E[G(z)])^2, \quad \text{where} \quad g(u) \quad = \frac{1}{a} \log \cosh(au), \text{ so } g'(u) = \tanh(au) \;\; 1 \leq a \leq 2$$

- ICA is quite tricky to implement due to numerical issues.

6.5 & 6.6 LLE & Isomap

- Both the following algorithums lower dimensions while preserving the neighbourhood relations in the data:
  1. Locally Linear Embedding (LLE) tries to approximate the data by sticking together sets of locally flat patches that cover the dataset

  2. Isomap uses the shortest distances (geodesics) on the non-linear space to find a globally optimal solution

6.5 LOCALLY LINEAR EMBEDDING (LLE)

- LLE Approximates the data by sticking together sets of locally flat patches that cover the dataset
- Idea: by making linear approximations we make some errors => so make these errors as small as possible by making the patches small where there is lots of non-linearity in the data.
- Reconstruction error => sum-of-squares of the distance

$$\epsilon = \sum_{i=1}^{N} \left(\mathbf{x}_i - \sum_{j=1}^{N} \mathbf{W}_{ij} \mathbf{x}_j \right)^2 .$$

- Wij say how much effect the jth datapoint has on the reconstruction of the ith point

6.5 LOCALLY LINEAR EMBEDDING (LLE) continued…

- If another point is a long way off, probably isn't useful, so only use points that are close (in its neighbourhood)
- Neighbourhood methods:
  - Points that are less than some predefined distance d to the current point are neighbours (don't know how many neighbours, but they are all close)
  - The k nearest points are neighbours (know how many, but some could be far)
- Solving Wij is a least-squares problem => simplified by enforcing neighbourhood constraints (if xj far from xi, Wij = 0) and sumj(Wij) = 1
- So far we reconstruct but don't reduce dimensions. To reduce dimensions using same formula but minimising distance yi in some lower dimensional space L.

$$\mathbf{y}_i = \sum_{i=1}^{N} \left(\mathbf{y}_i - \sum_{j=1}^{L} \mathbf{W}_{ij} \mathbf{y}_j \right)^2 .$$

- Solving this, the solution is the eigenvalues of the quadratic form matrix:

$\mathbf{M}_{ij} = \delta_{ij} - \mathbf{W}_{ij} - \mathbf{W}_{ji} + \sum_k \mathbf{W}_{ji}\mathbf{W}_{kj}$, where δij is the Kronecker delta function, so δij = 1 if i = j and 0 otherwise

The Locally Linear Embedding Algorithm

- Decide on the neighbours of each point (e.g., K nearest neighbours):
  - compute distances between every pair of points
  - find the k smallest distances
  - set $\mathbf{W}_{ij} = 0$ for other points
  - for each point $\mathbf{x}_i$:
    - create a list of its neighbours' locations $\mathbf{z}_i$
    - compute $\mathbf{z}_i = \mathbf{z}_i - \mathbf{x}_i$
- Compute the weights matrix $\mathbf{W}$ that minimises Equation (6.31) according to the constraints:
  - compute local covariance $\mathbf{C} = \mathbf{Z}\mathbf{Z}^T$, where $\mathbf{Z}$ is the matrix of $\mathbf{z}_i$s
  - solve $\mathbf{C}\mathbf{W} = \mathbf{I}$ for $\mathbf{W}$, where $\mathbf{I}$ is the $N \times N$ identity matrix
  - set $\mathbf{W}_{ij} = 0$ for non-neighbours
  - set other elements to $\mathbf{W}/\sum(\mathbf{W})$
- Compute the lower dimensional vectors $\mathbf{y}_i$ that minimise Equation (6.32):
  - create $\mathbf{M} = (\mathbf{I} - \mathbf{W})^T(\mathbf{I} - \mathbf{W})$
  - compute the eigenvalues and eigenvectors of $\mathbf{M}$
  - sort the eigenvectors into order by size of eigenvalue
  - set the qth row of $\mathbf{y}$ to be the $q+1$ eigenvector corresponding to the qth smallest eigenvalue (ignore the first eigenvector, which has eigenvalue 0)

- Unrolls swissroll

6.6 ISOMAP & Multi-Dimensional Scaling (MDS)

- ISOMAP tries to minimise the global error by looking at all of the pairwise distances and computing global geodesics.
- It’s a variant of the standard multi-dimensional scaling (MDS) so we explain MDS first.

MDS

- Like PCA, MDS tries to find a linear approximation in a lower dimensionality.
- MDS tries to preserve the distances between all pairs of points.
- Does this by minimising a cost function using gradient descent between **x** with M dimensions and **z** with L dimensions where L < M
- Cost function:
  - Kruskal–Shephard scaling (also known as least-squares)

$$S_{KS}(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N) = \sum_{i \neq i'} (d_{ii'} - \|\mathbf{z}_i - \mathbf{z}'_i\|)^2$$

  - Or Sammon mapping

$$S_{SM}(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N) = \sum_{i \neq i'} \frac{(d_{ii'} - \|\mathbf{z}_i - \mathbf{z}'_i\|)^2}{d_{ii'}}$$

(More weight on short distances, neighbouring points stay the correct distance apart)

Classical MDS

- Classical MDS uses similarities between datapoints rather than distances
- Similarities constructed with centred inner product $s_{ii'} = (\mathbf{x}_i - \bar{\mathbf{x}}), (\mathbf{x}'_i - \bar{\mathbf{x}})^T$
- Direct algorithm that does not have to use gradient descent by minimising $\sum_{i \neq i'} \left(s_{ii'} - (\mathbf{z}_i - \bar{\mathbf{z}}), (\mathbf{z}'_i - \bar{\mathbf{z}})^T \right)^2$

The Multi-Dimensional Scaling (MDS) Algorithm

- Compute the matrix of squared pairwise similarities $\mathbf{D}$, $\mathbf{D}_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$
- Compute $\mathbf{J} = \mathbf{I}_N - 1/N$ (where $\mathbf{I}_N$ is the $N \times N$ identity function and N is the number of datapoints)
- Compute $\mathbf{B} = -\frac{1}{2}\mathbf{J}\mathbf{D}\mathbf{J}^T$
- Find the L largest eigenvalues λ_i of $\mathbf{B}$, together with the corresponding eigenvectors $\mathbf{e}_i$
- Put the eigenvalues into a diagonal matrix $\mathbf{V}$ and set the eigenvectors to be columns of matrix $\mathbf{P}$
- Compute the embedding as $\mathbf{X} = \mathbf{P}\mathbf{V}^{1/2}$

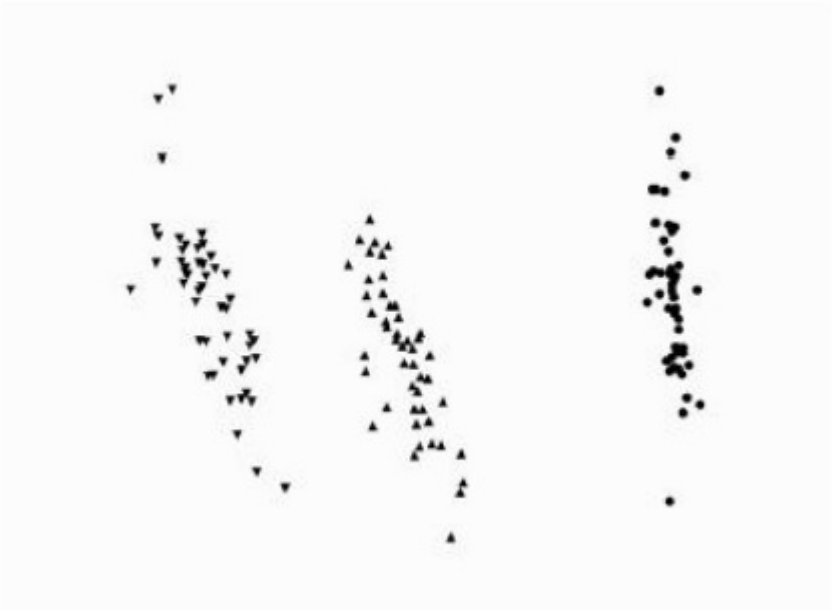
Isomap

- This classical MDS algorithm works fine on flat manifolds (dataspaces)
- Isomap works on manifolds that are not flat

The Isomap Algorithm

- Construct the pairwise distances between all pairs of points
- Identify the neighbours of each point to make a weighted graph G
- Estimate the geodesic distances d_G by finding shortest paths
- Apply classical MDS to the set of d_G

Goal is to constructs estimated distance matrix for all pairs of datapoints on the manifold without information about the manifold.
For close points, assuming distances between pairs of points are good (neighbours).
For “far” points, builds up the distances by finding paths that run through points that are close together, i.e., that are neighbour.
Algorithm then uses MDS on this distance matrix.



Iris data set: neighbourhood size > 50 to ensure path to “far” points through neighbours so points don’t become disconnected

Good swissroll too