`tapkee` methods

Contents

1 Diffusion map

The diffusion map algorithm performs the following steps to embed feature vectors $x_1, \ldots, x_N$:

- Compute $N \times N$ gaussian kernel matrix K such that

$$K_{i,j} = exp\left\{-\frac{d^2(x_i, x_j)}{\omega}\right\},$$

  where $d : X \times X \to \mathbb{R}$ is a distance function and $\omega > 0$ is a width of the kernel.

- Transform the matrix K using the following equations

$$K_{i,j} \leftarrow \frac{K_{i,j}}{(p_i p_j)^q},$$

  where $p_i = \sum_{j=1}^{N} K_{j,i}$. Only $q = 1$ for 'standard' diffusion map is currently supported. Then, recompute $p_i = \sum_{j=1}^{N} K_{j,i}$ again and do

$$K_{i,j} \leftarrow \frac{K_{i,j}}{\sqrt{p_i p_j}}.$$

- Construct embedding with dim $= d$ from the solution of the following partial eigenproblem

$$Kf = \lambda f$$

  for $d + 1$ largest eigenvalues. Form the embedding matrix such that the i-th coordinate $(i = 1, \ldots, N)$ of j-th largest eigenvector $(j = 2, \ldots, d + 1)$ corresponds to j-th coordinate of projected i-th vector, normalized by λ_i^t and the first eigenvector corresponding to $\lambda_1 = 1$.

1.1 References

- Coifman, R., & Lafon, S. (2006). Diffusion maps

2 Factor analysis

Factor analysis aims at describing how several observed variables are correlated to each other by means of identifying a set of unobserved variables, the so-called factors. Desirably, the number of factors is shorter than the number of observed variables.

Factor analysis is an iterative algorithm. First of all the projection matrix is initialized randomly and the factors variance is set to the identity. Then, every iteration consists of the following steps:

- Compute the regularized inverse covariance matrix of the projection.
- Update the factors variance matrix.
- Update the projection matrix.
- Check for convergence using the log-likelihood of the model. If the difference between the current log-likelihood and the previous iteration's log-likelihood is below a threshold, then the algorithm has converged.

2.1 References

- Spearman, C. (1904). General Intelligence, Objectively Determined and Measured.

3 Hessian Locally Linear Embedding

Just like the Local Tangent Space Alignment, the Hessian Locally Linear Embedding algorithm is very similar to the Locally Linear Embedding algorithm.

Given a set of feature vectors $X = \{x_1, x_2, \dots x_N\}$ the HLLE algorithm proposes to perform the following steps:

- Identify nearest neighbors. For each $x \in X$ identify its k nearest neighbors, i.e. a set $\mathcal{N}_x$ of k feature vectors such that
$$\arg\min_{\mathcal{N}_x} \sum_{x_n \in \mathcal{N}_x} \|x - x_n\|_2^2$$
- Analyze hessian of each local patch. For each $x \in X$ compute the Gram matrix G of its neighbors such that $G_{i,j} = (\mathcal{N}_x^i, \mathcal{N}_x^j)$ and center it. Compute its t (the number of required features) eigenvectors $v_1, \dots v_t$. Construct hessian approximating matrix
$$Y = \begin{bmatrix} 1_k & v_1 & \dots & v_t & v_1 \cdot v_1 & \dots & v_1 \cdot v_t & \dots \end{bmatrix},$$
where $\cdot : X \times X \to X$ denotes coefficient-wise product. Normalize columns of the matrix Y and then compute matrix
$$Q = YY^T$$
and put it to the sparse alignment matrix L (initially set by zeroes) using the following procedure:
$$L \leftarrow L + Q.$$
- Embedding through eigendecomposition. To obtain t features (coordinates) of embedded vectors solve the partial eigenproblem
$$Lf = \lambda f,$$
for smallest eigenvalues $\lambda_1, \dots, \lambda_t, \lambda_{t+1}$ and its corresponding eigenvectors $f_1, \dots, f_t, f_{t+1}$. Drop the smallest eigenvalue $\lambda_1 \sim 0$ (with its corresponding eigenvector) and form embedding matrix such that i-th coordinate ($i = 1, \dots, N$) of j-th eigenvector ($j = 1, \dots, t$) corresponds to j-th coordinate of projected i-th vector.

3.1 References

- Donoho, D., & Grimes, C. (2003). Hessian eigenmaps: new locally linear embedding techniques for high-dimensional data

4 Isomap

The Isomap algorithm can be considered as a modification of the classic Multidimensional Scaling algorithm. The algorithm itself consists of the following steps:

- For each feature vector $x \in X$ find k its nearest neighbors and construct the sparse neighborhood graph.
- Compute squared distances matrix D such as $D_{i,j} = d^2(x_i, x_j)$.
- Relax distances with shortest (so-called geodesic) distances on the sparse neighborhood graph (e.g. with Dijkstra's algorithm).
- Center the matrix D with subtracting row mean, column mean and adding the grand mean. Multiply D element-wise with -0.5.
- Compute embedding with the t eigenvectors that correspond to the largest eigenvalues of the matrix D; normalize these vectors with dividing each eigenvector by the square root of its corresponding eigenvalue. Form the final embedding with eigenvectors as rows and projected feature vectors as columns.

4.1 References

- Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction

5 Kernel Principal Component Analysis

The Kernel Principal Component Analysis algorithm is a generalization of the PCA algorithm. The algorithm performs the following steps

- Compute the kernel matrix K such that $K_{i,j} = k(x_i, x_j)$ where $k : X \times X \to \mathbb{R}$ is a Mercer kernel function and X is a set of feature vectors
  $x_1, x_2, \ldots, x_N$
- Center the matrix K with subtracting row mean, column mean and adding the grand mean.
- Compute embedding with the t eigenvectors that correspond to the largest eigenvalues of the matrix D; normalize these vectors with dividing each eigenvectors with square root of its corresponding eigenvalue. Form the final embedding with eigenvectors as rows and projected feature vectors as columns.

5.1 References

- Schölkopf, B., Smola, A., & Müller, K. R. (1997). Kernel principal component analysis

6 Laplacian Eigenmaps

The Laplacian Eigenmaps algorithm performs the following simple steps to embed given feature vectors $x_1, \ldots, x_N$:

- Identify nearest neighbors. For each $x \in X$ identify its k nearest neighbors, i.e. a set $\mathcal{N}_x$ of k feature vectors such that
$$\arg\min_{\mathcal{N}_x} \sum_{x_n \in \mathcal{N}_x} d(x, x_n),$$
where $d : X \times X \to \mathbb{R}$ is a distance function.

- Construct weight matrix. Initially setting $N \times N$ matrix W to zero, set
$$W_{i,j} = exp\left\{-\frac{d^2(x_i, x_j)}{\tau}\right\}$$
iff for i-th vector x_i neighbors set $\mathcal{N}_{x_i}$ contains x_j and vice versa (so-called mutual neighborhood). Find a diagonal matrix D such that $D_{i,i} = \sum_{j=1}^{N} W_{j,i}$.

- Find embedding throught eigendecomposition. To obtain t features (coordinates) of embedded vectors solve the partial generalized eigenproblem
$$(D - W)f = \lambda D f,$$
for smallest eigenvalues $\lambda_1, \ldots, \lambda_t, \lambda_{t+1}$ and its corresponding eigenvectors $f_1, \ldots, f_t, f_{t+1}$. Drop the smallest eigenvalue $\lambda_1 \sim 0$ (with the corresponding eigenvector) and form embedding matrix such that i-th coordinate ($i = 1, \ldots, N$) of j-th eigenvector ($j = 1, \ldots, t$) corresponds to j-th coordinate of projected i-th vector.

6.1 References

- Belkin, M., & Niyogi, P. (2002). Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering

7 Locally Linear Embedding

Given a set of feature vectors $X = \{x_1, x_2, \ldots x_N\}$ the Locally Linear Embedding algorithm proposes to perform the following steps:

- Identify nearest neighbors. For each $x \in X$ identify its k nearest neighbors, i.e. a set $\mathcal{N}_x$ of k feature vectors such that
$$\arg\min_{\mathcal{N}_x} \sum_{x_n \in \mathcal{N}_x} \|x - x_n\|_2^2$$

- Compute linear reconstruction weights. For each $x \in X$ compute weight vector $w \in \mathbb{R}^n$ that minimizes
$$\|x - \sum_{i=1}^{k} w_i \mathcal{N}_x^i\|_2, \quad \text{w.r.t.} \quad \|w\|_2 = 1$$

where $\mathcal{N}_x^i$ is a i-th element of the set $\mathcal{N}_x$. The solution of the problem stated above can be found from the normalized solution of the following equation:

$$Gw = 1_k,$$

where G is a $k \times k$ matrix such that $G_{i,j} = (x - \mathcal{N}_x^i)(x - \mathcal{N}_x^j)$ and $1_k \in \mathbb{R}^k$ is a vector of all ones. Obviously, the problem comes ill-posed in case k gets more than dimension of feature space X. This can be avoided with the regularization:

$$G \leftarrow G + \varepsilon E,$$

where E is an identity matrix and ε is a pre-defined constant reconstruction shift (usually 10^{-3}). Once w is computed it is stored into the sparse alignment matrix L (initially set by zero) with the following procedure:

$$L_{I,I} \leftarrow L_{I,I} + W,$$

where I is a set containing indices of all element of the set $\mathcal{N}_x$ and x itself, $L_{I,I}$ denotes all (i, j) elements of the sparse matrix L such that $i, j \in I$ and

$$W = \begin{bmatrix} 1 & -w \\ -w^T & w^T w \end{bmatrix}$$

.

- Embedding through eigendecomposition. To obtain t features (coordinates) of embedded vectors solve the partial eigenproblem

$$Lf = \lambda f,$$

for smallest eigenvalues $\lambda_1, \ldots, \lambda_t, \lambda_{t+1}$ and its corresponding eigenvectors $f_1, \ldots, f_t, f_{t+1}$. Drop the smallest eigenvalue $\lambda_1 \sim 0$ (with the corresponding eigenvector) and form embedding matrix such that i-th coordinate $(i = 1, \ldots, N)$ of j-th eigenvector $(j = 1, \ldots, t)$ corresponds to j-th coordinate of projected i-th vector.

8 Locally Linear Embedding

The Locally Linear Embedding algorithm can be generalized for spaces with defined dot product function $k(x, y)$ (so-called RKHS) in the elegant way. Using the following equation

$$||x - y||_2^2 = (x, x) - 2(x, y) + (y, y)$$

we may transform the nearest neighbors problem to the following form:

$$\arg\min_{\mathcal{N}_x} \sum_{x_n \in \mathcal{N}_x} \left[k(x, x) - 2k(x, x_n) + k(x_n, x_n) \right].$$

The matrix G can be formulated in terms of dot product as well. To find G using only dot products we can compute the Gram matrix K such that $K_{i,j} = k(x_i, x_j)$ and center it using the matrix $C_k = E_k - \frac{1}{k} 1 1^T$:

$$G = K C_k K.$$

There is an efficient way to compute that - it is can be done with subtracting a column mean of K from each column of K, subtracting a row mean of K from each row of K and adding the grand mean of all elements of K to K.

8.1 References

- Sam Roweis' page on LLE
- Saul, L. K., Ave, P., Park, F., & Roweis, S. T. (2001). An introduction to Locally Linear Embedding
- Zhao, D. (2006) Formulating LLE using alignment technique

9 Linear Local Tangent Space Alignment

The Linear Local Tangent Space Alignment is a modification of the LTSA algorithm. Main difference (just like in NPE and LLE) of linear and original LTSA methods lies in the way of constructing embedding. Instead of solving common for LLE and LTSA eigenproblem, LLTSA requires solving the following generalized eigenproblem:

$$RLR^T f = \lambda RR^T f,$$

where R is a matrix containing all feature vectors $x_1, \ldots, x_N$ row-wise. The problem is solved for smallest eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_t$ and its corresponding eigenvectors $f_1, \ldots, f_t$. To find final embedding LLTSA forms a matrix such that i-th coordinate ($i = 1, \ldots, N$) of j-th eigenvector ($j = 1, \ldots, t$) corresponds to j-th coordinate of projected i-th vector.

9.1 References

- Zhang, T., Yang, J., Zhao, D., & Ge, X. (2007). Linear local tangent space alignment and application to face recognition

10 Locality Preserving Projections

The Locality Preserving Projections algorithm can be viewed as a linear approximation of the Laplacian Eigenmaps algorithm. It reproduces first two steps of the Laplacian Eigenmaps and the difference lies in the step 3. To obtain t features (coordinates) of embedded vectors LPP solves the partial generalized eigenproblem

$$R(D - W)R^T f = \lambda RDR^T f,$$

where R contains all feature vectors row-wise, for smallest eigenvalues $\lambda_1, \ldots, \lambda_t, \lambda_{t+1}$ and its corresponding eigenvectors $f_1, \ldots, f_t, f_{t+1}$. Drop the smallest eigenvalue $\lambda_1 \sim 0$ (with the corresponding eigenvector) and form embedding matrix such that i-th coordinate ($i = 1, \ldots, N$) of j-th eigenvector ($j = 1, \ldots, t$) corresponds to j-th coordinate of projected i-th vector.

10.1 References

- He, X., & Niyogi, P. (2003). Locality Preserving Projections

11 Local Tangent Space Alignment

The Local Tangent Space Alignment algorithm is pretty similar to the Locally Linear Embedding algorithm.

Given a set of feature vectors $X = \{x_1, x_2, \ldots x_N\}$ the Local Tangent Space Alignment algorithm performs the following steps:

- Identify nearest neighbors. For each $x \in X$ identify its k nearest neighbors, i.e. a set $\mathcal{N}_x$ of k feature vectors such that

$$\arg\min_{\mathcal{N}_x} \sum_{x_n \in \mathcal{N}_x} \|x - x_n\|_2^2$$

- Perform principal component analysis of each local neighborhood patch. For each $x \in X$ compute the Gram matrix G of its neighbors such that $G_{i,j} = (\mathcal{N}_x^i, \mathcal{N}_x^j)$ and center it. Compute its t (the number of required features) eigenvectors and store it in the matrix V. Compute matrix

$$Q = \begin{bmatrix} 1_k & V \end{bmatrix} \begin{bmatrix} 1_k \\ V \end{bmatrix}$$

  and put it to the sparse alignment matrix L (initially set by zeroes) using the following procedure:

$$L \leftarrow L + E_k - Q.$$

- Embedding through eigendecomposition. To obtain t features (coordinates) of embedded vectors solve the partial eigenproblem

$$Lf = \lambda f,$$

  for smallest eigenvalues $\lambda_1, \ldots, \lambda_t, \lambda_{t+1}$ and its corresponding eigenvectors $f_1, \ldots, f_t, f_{t+1}$. Drop the smallest eigenvalue $\lambda_1 \sim 0$ (with the corresponding eigenvector) and form embedding matrix such that i-th coordinate ($i = 1, \ldots, N$) of j-th eigenvector ($j = 1, \ldots, t$) corresponds to j-th coordinate of projected i-th vector.

12 Kernel Local Tangent Space Alignment

Like the Locally Linear Embedding algorithm, LTSA allows generalization for Mercer kernel functions. Nearest neighbors computation in KLTSA is identical to one in KLTSA and the matrix G in the step 2 is naturally replaced with matrix $K_{i,j} = k(\mathcal{N}_x^i, \mathcal{N}_x^j)$.

12.1 References

- Zhang, Z., & Zha, H. (2002). Principal Manifolds and Nonlinear Dimension Reduction via Local Tangent Space Alignment

13 Classic multidimensional scaling

The classic multidimensional scaling algorithm is probably the simplest dimensionality reduction algorithm which reduced data in an attempt to keep pairwise distances the same. The algorithm itself is:

- For a given set of vectors $X = x_1, x_2, \ldots, x_N$ compute the pairwise distances matrix D such that $D_{i,j} = d(x_i, x_j)$,
- Square each element of the distances matrix D and center the matrix with subtracting row mean, column mean and adding the grand mean.

- Compute embedding with the t eigenvectors that correspond to the largest eigenvalues of the matrix D; normalize these vectors with dividing each eigenvectors with square root of its corresponding eigenvalue. Form the final embedding with eigenvectors as rows and projected feature vectors as columns.

14 Neighborhood Preserving Embedding

The Neighborhood Preserving Embedding (NPE) algorithm can be considered as a linear approximation of the Locally Linear Embedding algorithm. Thus most of computation routines can be shared with LLE. The NPE algorithm uses steps 1 and 2 of the Locally Linear Embedding and the main difference lies in the eigendecomposition based embedding.

According to the NPE algorithm embedding can be found from the solution of the following partial generalized eigenproblem:

$$RLRf = \lambda RR^T f$$

where R is a matrix containing all feature vectors $x_1, \ldots, x_N$ row-wise. The problem is solved for smallest eigenvalues $\lambda_1, \ldots, \lambda_t$ and its corresponding eigenvectors $f_1, \ldots, f_t$. The final embedding is obtained with a matrix such that i-th coordinate ($i = 1, \ldots, N$) of j-th eigenvector ($j = 1, \ldots, t$) corresponds to j-th coordinate of projected i-th vector.

References

- He, X., Cai, D., Yan, S., & Zhang, H.-J. (2005). Neighborhood preserving embedding

15 Principal Component Analysis

The Principal Component Analysis is probably the oldest dimension reduction algorithm which comes in various flavours today. The simplest 'version' of the PCA algorithm could look like that:

- Subtract mean feature vector from each feature vector of a set $X = \{x_1, x_2, \ldots, x_N\}$.
- Compute the covariance matrix C using all feature vectors.
- Find top t (desired dimension of embedded space) and form projection matrix P with eigenvectors as columns.
- Project the data with $Y = PX$.

16 Random projection

The Random projection algorithm is yet more simple algorithm (comparing to PCA and MDS). It can be said that the algorithm is based on Johnson-Lindenstrauss lemma that states that a small number of vectors in high-dimensional space can be embedded into a space of much lower dimension with keeping pairwise distances nearly preserved. The algorithm itself is:

- Construct random basis matrix P with normalized random gaussian vectors as columns.
- Project data with left multiplication with generated matrix $Y = PX$.

17 Stochastic Proximity Embedding

Stochastic Proximity Embedding (SPE) acts on a set of N vectors $Y = \{y_1, y_2, \dots y_N\}$ with corresponding symmetric distance matrix D_{ij} in the following manner:

1. Choose an initial learning rate λ.
2. Initialize randomly the point coordinates in the embedded space $X = \{x_1, x_2, \dots x_N\}$.
3. Select at random a pair of points with indices i and j. For a prescribed number of iterations S, compute their distances in the embedded space,

   $$d_{i,j} = \|x_i - x_j\|$$

   ; if $d_{i,j} \neq D_{i,j}$ then update the coordinates of the selected points by

   $$x_i \leftarrow x_i + \lambda \frac{1}{2} \frac{D_{ij} - d_{ij}}{d_{ij} + \epsilon}(x_i - x_j),$$

   $$x_j \leftarrow x_j + \lambda \frac{1}{2} \frac{D_{ij} - d_{ij}}{d_{ij} + \epsilon}(x_j - x_i).$$

4. Decrease the learning rate λ by $\delta\lambda | 0 < \delta < 1$. λ is decreased to avoid oscillatory behaviour.
5. Repeat steps 3 and 4 for a predetermined number of iterations C.

SPE is an interesting method because of its simplicity and efficiency, as it scales linearly with the sample size N.

17.1 Reference

- D. K. Agrafiotis. "Stochastic Proximity Embedding," *Journal of Computational Chemistry*, 2003.

18 Stochastic Neighbour Embedding

Stochastic Neighbour Embedding (SNE) uses conditional probability densities in order to model pairwise similarities between data points, rather than using Euclidean distances directly. The similarity of the point x_j to the point x_i is the conditional probability $p_{j|i}$, which is the probability that x_j would be x_i's neighbour taking into account that neighbourhoods are built in proportion to Gaussian probability densities centered at x_i. Formally,

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

where σ_i is the variance of the Gaussian centered on x_i, whose computation will be later explained. The similarities in the low-dimensional space are defined in a similar way. However, the variance of the Gaussian distributions employed are fixed to $\frac{1}{\sqrt{2}}$ this time, i.e.

$$q_{j|i} = \frac{exp(-\|x_i - x_j\|^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2)}.$$

Intuitively, if the low-dimensional points Y_i and Y_j map correctly the similarity between their high-dimensional counterparts x_i and x_j, then $p_{j|i}$ and $q_{j|i}$ will be close to each other. SNE aims at making these quantities as close as possible minimizing the sum of Kullback-Leibler divergences over all the data set. Thus, the cost function is

$$C = \sum_i KL(P_i \| Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}.$$

Unfortunately, there exists no optimal value of the Gaussian variance for all the points in the data set since the data may vary considerably throughout the data set. SNE chooses the value of each $sigma_i$ performing binary search so that a user specified value for the Shannon entropy of P_i is achieved.

19 t-Distributed Stochastic Neighbour Embedding

There are two main issues related to SNE that t-Distributed Stochastic Neighbour Embedding (t-SNE) addresses:

- SNE's cost function using gradient descent is faster optimized using symmetric similarities. Therefore, t-SNE uses joint probability distributions p_{ij} and q_{ij} instead of conditional distributions.
- t-SNE uses Student's t instead of Gaussian distributions to handle better the so-called crowding problem.

19.1 References

- Van der Maaten, L., Hinton, G. (2008). Visualizing Data using t-SNE. 3