Optimizations:

High-dimensional datasets can be very difficult to visualize. While data in two or three dimensions can be plotted to show the inherent structure of the data, equivalent high-dimensional plots are much less intuitive. To aid visualization of the structure of a dataset, the dimension must be reduced in some way.

To address this concern, a number of supervised and unsupervised linear dimensionality reduction frameworks have been designed, such as Principal Component Analysis (PCA), Independent Component Analysis, Linear Discriminant Analysis, and others. These algorithms define specific rubrics to choose an "interesting" linear projection of the data. These methods can be powerful, but often miss important non-linear structure in the data.

Manifold Learning can be thought of as an attempt to generalize linear frameworks like PCA to be sensitive to non-linear structure in data. Though supervised variants exist, the typical manifold learning problem is unsupervised: it learns the high-dimensional structure of the data from the data itself, without the use of predetermined classifications.

I.) Manifold learning (generalize linear frameworks to be sensitive to non-linear structure in data):

Isomap:

Isomap can be viewed as an extension of Multi-dimensional Scaling (MDS) or Kernel PCA. Isomap seeks a lower-dimensional embedding which maintains geodesic distances between all points.

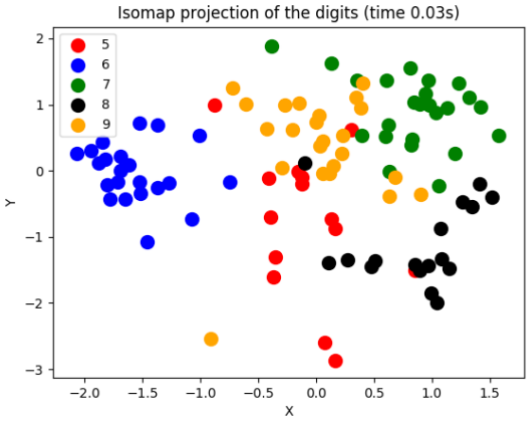
The Isomap algorithm comprises three stages:

1. Nearest neighbor search.
2. Shortest-path graph search.
3. Partial eigenvalue decomposition.

The overall complexity of Isomap is:

$O[D\log(k)N\log(N)] + O[N2(k + \log(N))]+O[dN2]$.

- N: number of training data points
- D: input dimension
- k: number of nearest neighbors
- d: output dimension



Local linear embedding:

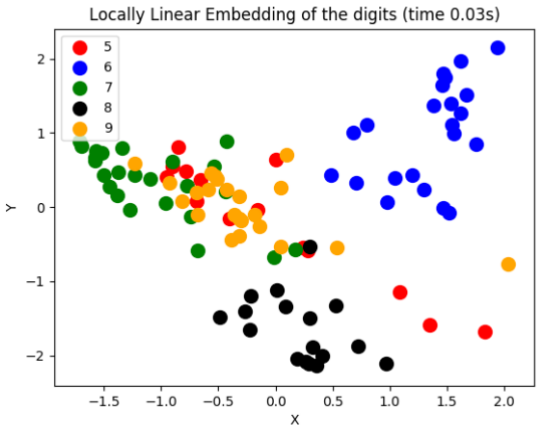
Locally linear embedding (LLE) seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods. It can be thought of as a series of local Principal Component Analyses which are globally compared to find the best non-linear embedding.

The standard LLE algorithm comprises three stages:

1. Nearest neighbor search.
2. Weight Matrix Construction.
3. Partial eigenvalue decomposition.

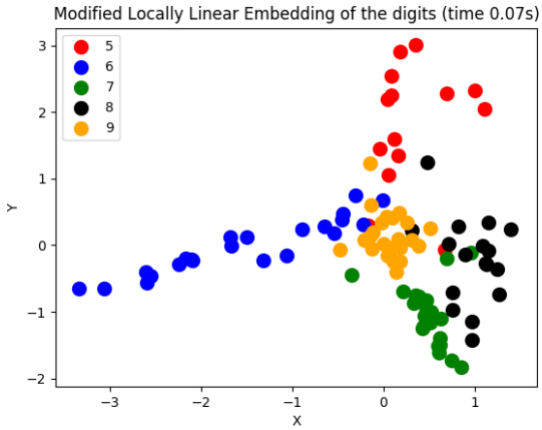
The overall complexity of standard LLE is:

$O[D\log(k)N\log(N)] + O[DNk3] + O[dN2]$.



Modified local linear embedding:

One well-known issue with LLE is the regularization problem. When the number of neighbors is greater than the number of input dimensions, the matrix defining each local neighborhood is rank-deficient. One method to address the regularization problem is to use multiple weight vectors in each neighborhood. This is the essence of *modified locally linear embedding* (MLLE).



The MLLE algorithm comprises three stages:

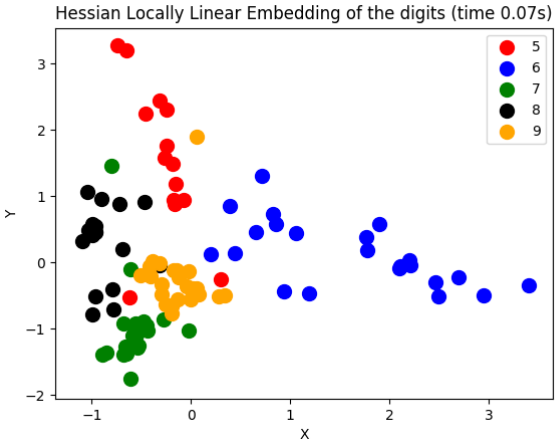
1. Nearest neighbor search.
2. Weight Matrix Construction.
3. Partial eigenvalue decomposition.

The overall complexity of MLLE is:

O[Dlog⁡(k)Nlog⁡(N)] + O[DNk3] + O[N(k−D)k2] + O[dN2].

Hessian eigenmapping:

Hessian Eigenmapping (known also as Hessian-based LLE) is another method of solving the regularization problem of LLE. It revolves around a hessian-based quadratic form at each neighborhood which is used to recover the locally linear structure. Though other implementations note its poor scaling with data size, `sklearn` implements some algorithmic improvements which make its cost comparable to that of other LLE variants for small output dimension.



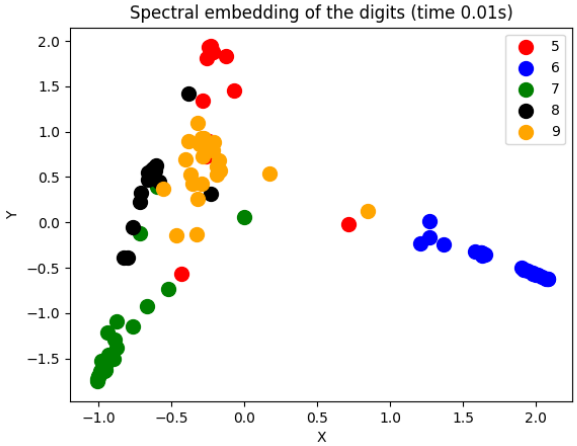
The HLLE algorithm comprises three stages:

1. Nearest neighbor search.
2. Weight Matrix Construction.
3. Partial eigenvalue decomposition.

The overall complexity of standard HLLE is:
O[Dlog⁡(k)Nlog⁡(N)] + O[DNk3] + O[Nd6] + O[dN2].

Spectral embedding:

Spectral Embedding is an approach to calculating a non-linear embedding. Scikit-learn implements Laplacian Eigenmaps, which finds a low dimensional representation of the data using a spectral decomposition of the graph Laplacian. The graph generated can be considered as a discrete approximation of the low dimensional manifold in the high dimensional space. Minimization of a cost function based on the graph ensures that points close to each other on the manifold are mapped close to each other in the low dimensional space, preserving local distances.

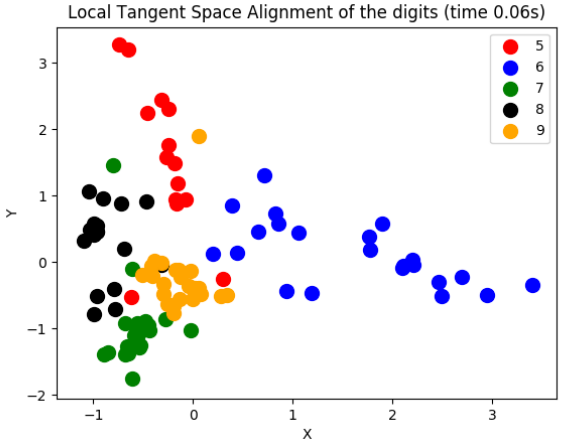


The Spectral Embedding (Laplacian Eigenmaps) algorithm comprises three stages:

1. Weight Graph Construction.
2. Graph Laplacian Construction.
3. Partial eigenvalue decomposition.

Local Tangent Space Alignment:

Though not technically a variant of LLE, Local tangent space alignment (LTSA) is algorithmically similar enough to LLE that it can be put in this category. Rather than focusing on preserving neighborhood distances as in LLE, LTSA seeks to characterize the local geometry at each neighborhood via its tangent space and performs a global optimization to align these local tangent spaces to learn the embedding.



The LTSA algorithm comprises three stages:

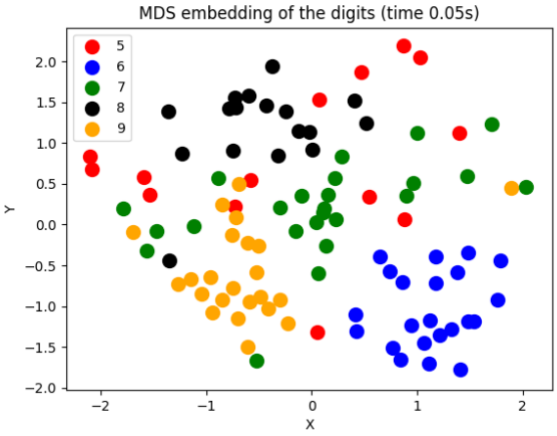
1. Nearest neighbor search.
2. Weight Matrix Construction.
3. Partial eigenvalue decomposition.

The overall complexity of standard LTSA is:
O[Dlog(k)Nlog(N)] + O[DNk3] + O[k2d] + O[dN2].

Multi-dimensional scaling (MDS):

Multi-dimensional scaling (MDS) seeks a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space. In general, is a technique used for analyzing similarity or dissimilarity data. MDS attempts to model similarity or dissimilarity data as distances in a geometric space.

There exists two types of MDS algorithm: metric and non metric. In the scikit-learn, the class MDS implements both. In Metric MDS, the input similarity matrix arises from a metric (and thus respects the triangular inequality), the distances between output two points are then set to be as close as possible to the similarity or dissimilarity data. In the non-metric version, the algorithms will try to preserve the order of the distances, and hence seek for a monotonic relationship between the distances in the embedded space and the similarities/dissimilarities.



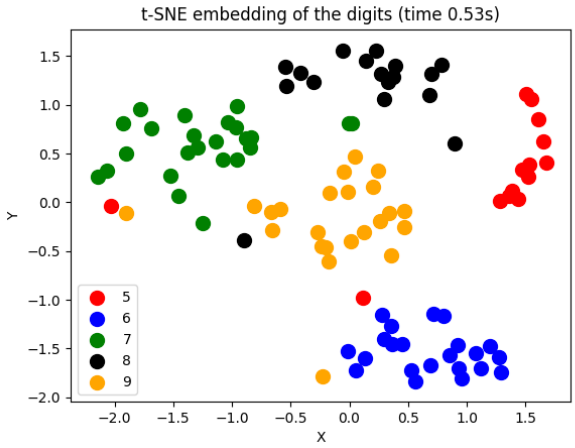
t-distributed Stochastic Neighbor Embedding (t-SNE):

t-SNE converts affinities of data points to probabilities. This allows t-SNE to be particularly sensitive to local structure and has a few other advantages over existing techniques:

- Revealing the structure at many scales on a single map.
- Revealing data that lie in multiple, different, manifolds or clusters.
- Reducing the tendency to crowd points together at the center.

While Isomap, LLE and variants are best suited to unfold a single continuous low dimensional manifold, t-SNE will focus on the local structure of the data and will tend to extract clustered local groups of samples as highlighted on the S-curve example. This ability to group samples based on the local structure might be beneficial to visually disentangle a dataset that comprises several manifolds at once as is the case in the digits dataset.

The disadvantages to using t-SNE are roughly:

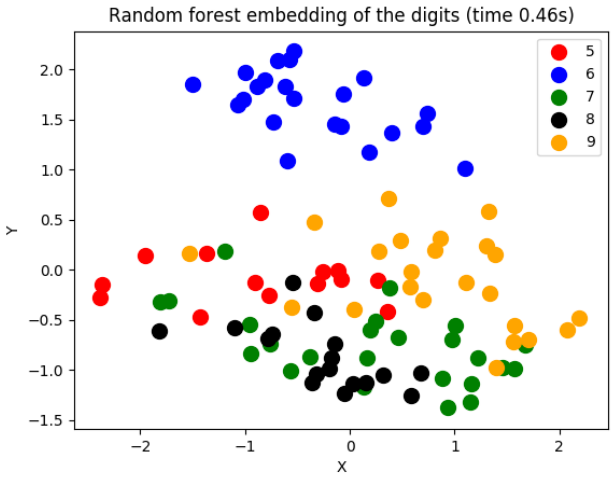


- t-SNE is computationally expensive, and can take several hours on million-sample datasets where PCA will finish in seconds or minutes
- The Barnes-Hut t-SNE method is limited to two or three dimensional embeddings.
- The algorithm is stochastic and multiple restarts with different seeds can yield different embeddings. However, it is perfectly legitimate to pick the embedding with the least error.
- Global structure is not explicitly preserved. This is problem is mitigated by initializing points with PCA (using init='pca').

Random forest embedding:

A forest embedding is a way to represent a feature space using a random forest. In our case, we are dealing with an unsupervised transformation of a dataset to a high-dimensional sparse representation. A data point is coded according to which leaf of each tree it is sorted into. Using a one-hot encoding of the leaves, this leads to a binary coding with as many ones as there are trees in the forest.

There may be a number of benefits in using forest-based embeddings:



1. **Distance calculations are ok when there are categorical variables:** as we're using leaf co-ocurrence as our similarity, we do not need to be concerned that distance is not defined for categorical variables.
2. **For supervised embeddings, we automatically set optimal weights for each feature for clustering:** if we want to cluster our data given a target variable, our embedding automatically selects the most relevant features.
3. **We do not need to worry about scaling features:** we do not need to worry about the scaling of the features, as we're using decision trees.

II.) PCA (linear framework not sensitive to non-linear structure in data):

Principal Component Analysis (PCA) is a dimension-reduction tool that can be used to reduce a large set of variables to a small set that still contains most of the information in the large set.

Objectives of principal component analysis:

1. PCA reduces attribute space from a larger number of variables to a smaller number of factors and as such is a "non-dependent" procedure (that is, it does not assume a dependent variable is specified).
2. PCA is a dimensionality reduction or data compression method. The goal is dimension reduction and there is no guarantee that the dimensions are interpretable (a fact often not appreciated by (amateur) statisticians).
3. To select a subset of variables from a larger set, based on which original variables have the highest correlations with the principal component.