Feature Selection - Feature Extraction

Vincent Barra

LIMOS, UMR 6158 CNRS, Université Clermont Auvergne

Why ?

Given some data, we often wish to perform preprocessing in order to:

- transform it to a format that our algorithms can take as input
- reduce time complexity: Less computation
- reduce space complexity: Less parameters
- make it have properties (0-mean, unit variance, sparseness,...)
- More interpretable: simpler explanation
- Data visualization (structure, groups, outliers, etc)

Curse of dimensionality

Assume n data lives in $[0, 1]^d$.

To capture a cube neighborhood representing a fraction s of $[0, 1]^d$, need the edge length to be $l = s^{1/d}$.

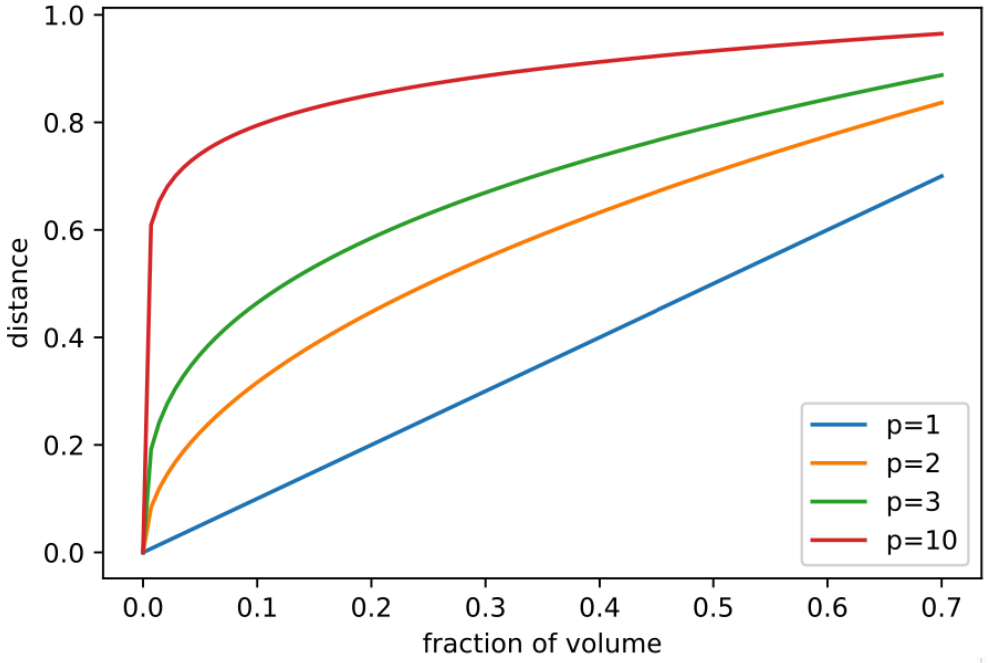
- d=10, s=0.1$\Rightarrow$ l=0.63
- d=10, s=0.01$\Rightarrow$ l=0.8

  $\Rightarrow$ Neighborhoods are no longer local

The volume V of an hypercube with an edge length l= 0.1 is V = 0.1^d

$\Rightarrow d \nearrow, V \searrow 0$

$\Rightarrow$ Probability to capture points becomes $\approx$ 0

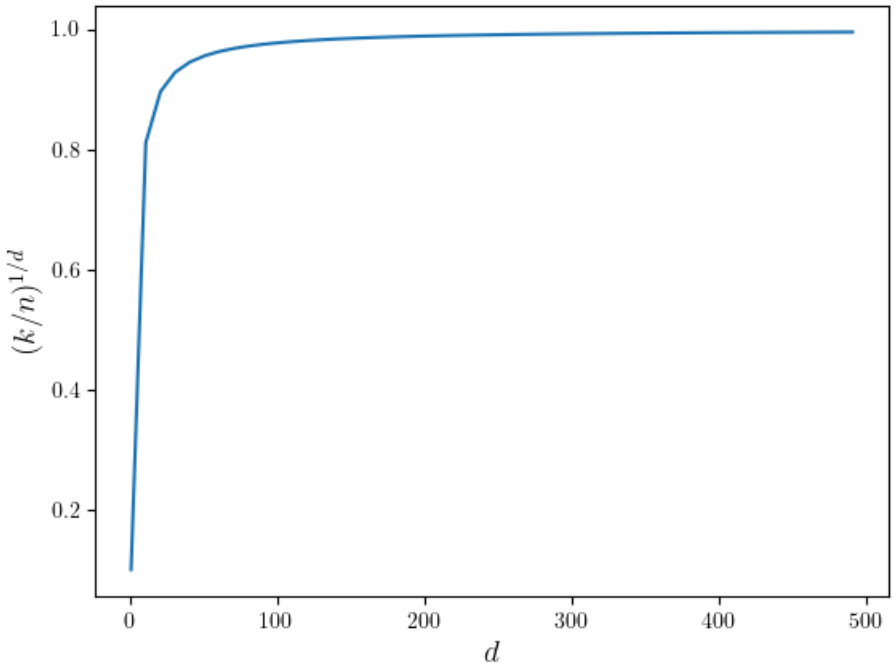


Curse of dimensionality

Points in high dimensional spaces are isolated

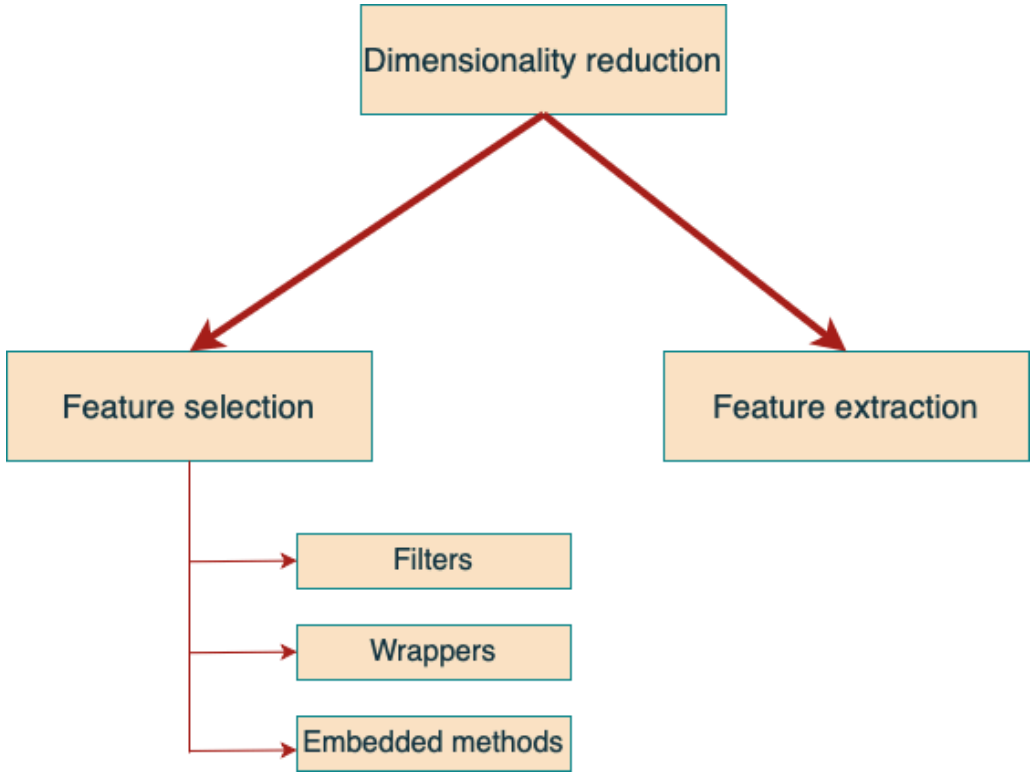
Immustration:

- Uniform sample n point in $[0, 1]^d$
- Compute the volume occupied by the k nearest neighbors of a point: $l^d \approx k/n$



Feature selection / extraction

- Feature selection
  - Choosing $k < d$ important features, ignoring the remaining $d - k$
    $\Rightarrow$ Subset selection algorithmss
- Feature extraction
  - Project the original d dimensions to **new** $k < d$ dimensions
    $\Rightarrow$ Discover low dimensional representations (smooth manifold) for data in high dimension.
    Manifold Learning



Implementation issues - scikit-learn

scikit learn Install User Guide API Examples Community More 1.5.1 (stable)

scikit-learn

Machine Learning in Python

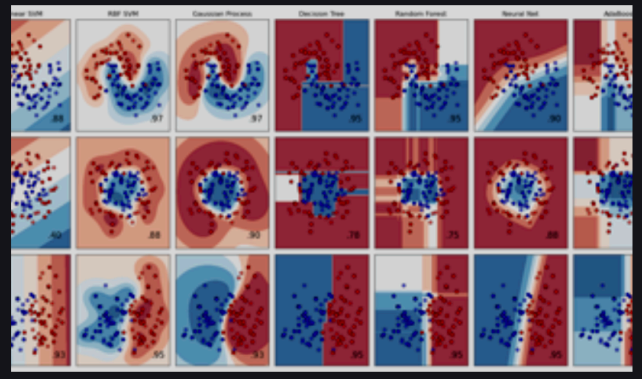
Getting Started | Release Highlights for 1.5

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

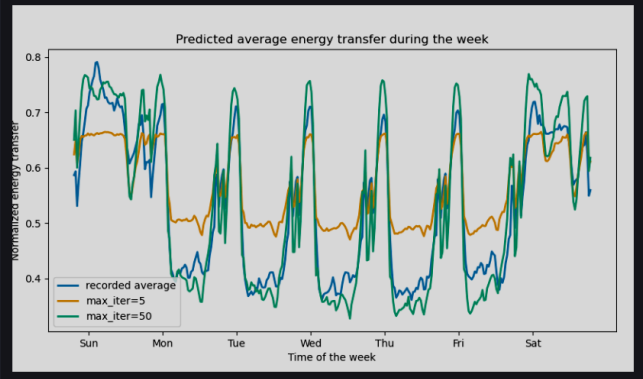


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

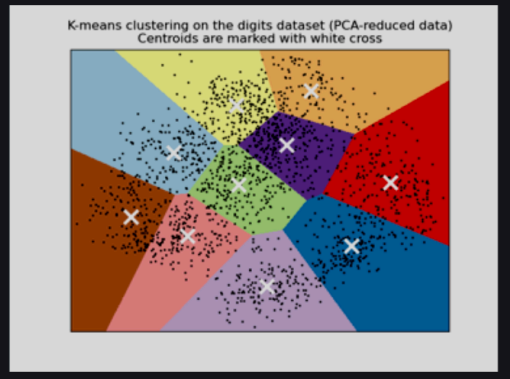


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...



Examples

Feature selection - Filter

- Variance threshold

  -Compute the variance of each feature

  - Assume that features with a higher variance may contain more useful information
  - Select the subset of features based on a user-specified threshold

👍 Good: fast!

👎 Does not take the relationship among features into account

```
from sklearn.feature_selection import VarianceThreshold
```

Feature selection - Filter

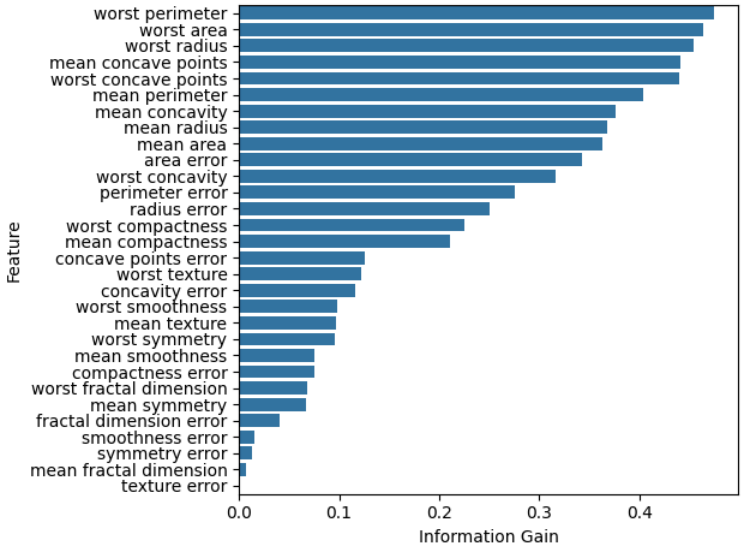
- Information gain
  - Measures the amount of information that feature f provides on dataset Z
  - Features with high information gain are more important
  - Based on the entropy

$$IG(Z, f) = H(Z) - H(Z|f)$$

$H(Z)$: entropy of the class distribution in Z

$H(Z|f)$: conditional entropy of the class distribution in Z given f

```
from sklearn.feature_selection import mutual_info_classif
```



Feature selection - Wrappers

Recursive Feature Elimination (RFE)

- Fit model to the training set (the model has to provide information about feature importance)
- Eliminate feature with the smallest coefficient
- Repeat until k features are reached

```
from sklearn.feature_selection import RFE
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
X, y = load_iris(return_X_y=True)
estimator = DecisionTreeClassifier()
s = RFE(estimator, n_features_to_select=2, step=1)
s.fit(X, y)

print("Taille des données avant sélection",X.shape)
print("Variables sélectionnées : ", s.get_support())
print("Classement des variables : ",s.ranking_)
```

```
Taille des données avant sélection (150, 4)
Variables sélectionnées :  [False False  True  True]
Classement des variables :  [2 3 1 1]
```

Feature selection - Wrappers

Permutation Importance

For each feature, do n times

- Shuffle feature column
- Assess performance w.r.t. original

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance
from sklearn.cluster import KMeans

data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)


model = KMeans(n_clusters=2).fit(X_train, y_train)

r = permutation_importance(model, X_test, y_test,n_repeats=30,random_state=0)

for i in r.importances_mean.argsort()[::-1]:
        print(f"{data.feature_names[i]:<8}"
              f"{r.importances_mean[i]:.3f}"
              f" +/- {r.importances_std[i]:.3f}")
```

```
petal length (cm)77.803 +/- 12.963
petal width (cm)39.936 +/- 6.399
sepal length (cm)30.218 +/- 6.479
sepal width (cm)5.483 +/- 1.083
```

Feature selection- Wrappers

Shapley values

Algorithm 6 (Estimation de la valeur de Shapley du descripteur i)

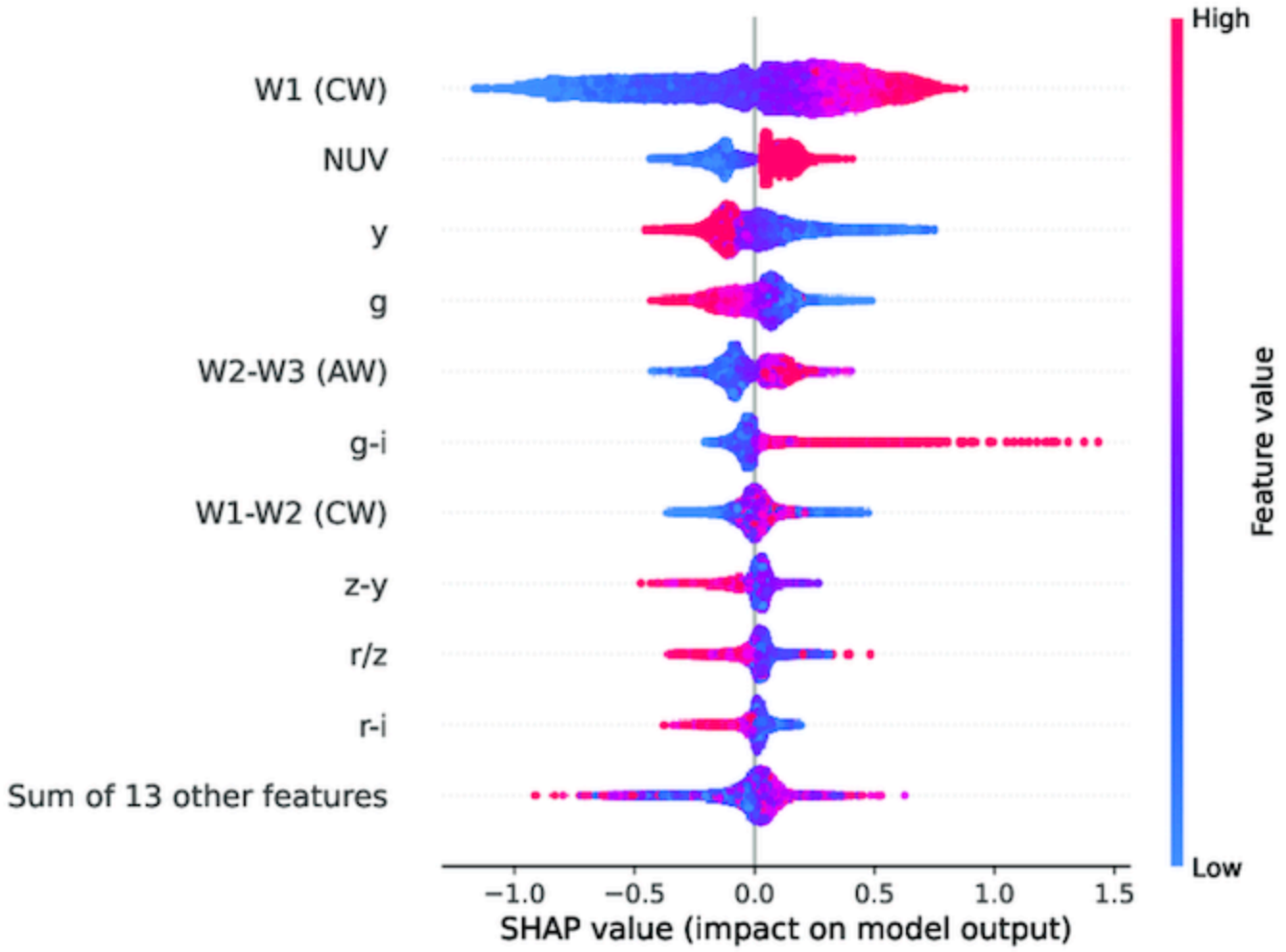
Entrée : Nombre d'itérations M, exemple $\mathbf{x}$, ensemble des exemples $\mathbf{X}$, i, modèle f

Sortie : Estimation de la valeur de Shapley du descripteur i

1. Pour tout $j \in [\![1, M]\!]$
    1. Tirer un exemple $\mathbf{z}$ dans $\mathbf{X}$
    2. Tirer une permutation aléatoire σ de l'ensemble $\{1 \cdots d\}$
    3. $\mathbf{x}_\sigma = (x_{\sigma(1)} \cdots x_{\sigma(d)})$ et $\mathbf{z}_\sigma = (z_{\sigma(1)} \cdots z_{\sigma(d)})$
    4. Créer deux nouveaux exemples :
        1. $\mathbf{x}_{+i} = (x_{\sigma(1)} \cdots x_{\sigma(i-1)}, x_{\sigma(i)}; z_{\sigma(i+1)} \cdots z_{\sigma(d)})$
        2. $\mathbf{x}_{-i} = (x_{\sigma(1)} \cdots x_{\sigma(i-1)}, z_{\sigma(i)}; z_{\sigma(i+1)} \cdots z_{\sigma(d)})$
    5. Calculer la contribution marginale du descripteur i : $\phi_i^j = f(\mathbf{x}_{+i}) - f(\mathbf{x}_{-i})$
2. Calculer un estimateur de la valeur de Shapley du descripteur i : $\phi_i(\mathbf{x}) = \frac{1}{M} \sum_{j=1}^{M} \phi_i^j$

Feature selection- Wrappers

Shapley values



Feature selection - Embedded

LASSO regularization

Least Absolute Shrinkage and Selection Operator

$$Min f_w(x) + \lambda \|w\|_1$$

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(penalty='l1')
```

Feature selection: Filters/Embedded methods

- Filters measure the relevance of features by their correlation with the dependent variable, while embedded methods measure the usefulness of a subset of features by training a model on them.
- Filters are much faster than enveloping methods
- Filters use statistical methods to evaluate a subset of features, while embedded methods use cross-validation.
- Filters may fail to find the best subset of features on many occasions, but enveloping methods can always provide the best subset of features.

Feature extraction

- Linear methods: PCA, MDS, LDA
- Non linear methods: ISOMAP, LLE, t-SNE

Linear Feature extraction - PCA

One standard method for decoupling and dimensionality reduction of continuous data is Principal Component Analysis (PCA)

- Find a low-dimensional space such that when x is projected there, information loss is minimized.
- The projection of x on the direction of w is: $z = w^T x$
- Find w such that $\mathbb{V}(z)$ is maximized

$$\begin{aligned}\mathbb{V}(z) &= \mathbb{V}(w^T x) = \mathbb{E}\left((w^T x - w^T \mu)^2\right) \\ &= \mathbb{E}\left((w^T x - w^T \mu)(w^T x - w^T \mu)\right) \\ &= \mathbb{E}\left(w^T (x-\mu)(x-\mu)^T w\right) \\ &= w^T \mathbb{E}\left((x-\mu)(x-\mu)^T\right) w \\ &= w^T \Sigma w\end{aligned}$$

Linear Feature extraction - PCA

- First PC*

  Maximize $\mathbb{V}(z)$ subject to $\|w\| = 1 \Rightarrow \max_u u^T \Sigma u - \alpha(u^T u - 1)$

  $\Sigma u = \alpha u \Rightarrow u$ eigenvector of Σ\

  Choose u with the largest eigenvalue for $\mathbb{V}(z)$ to be maximized

- Second PC*

  Deflation or $\mathbb{V}(z_2)$ subject to $\|w\| = 1$ and orthogonal to u

$$\max_w w^T \Sigma w - \alpha(w^T w - 1) - \beta(w^T u)$$

$\Sigma w = \alpha w \Rightarrow w$ eigenvector of Σ

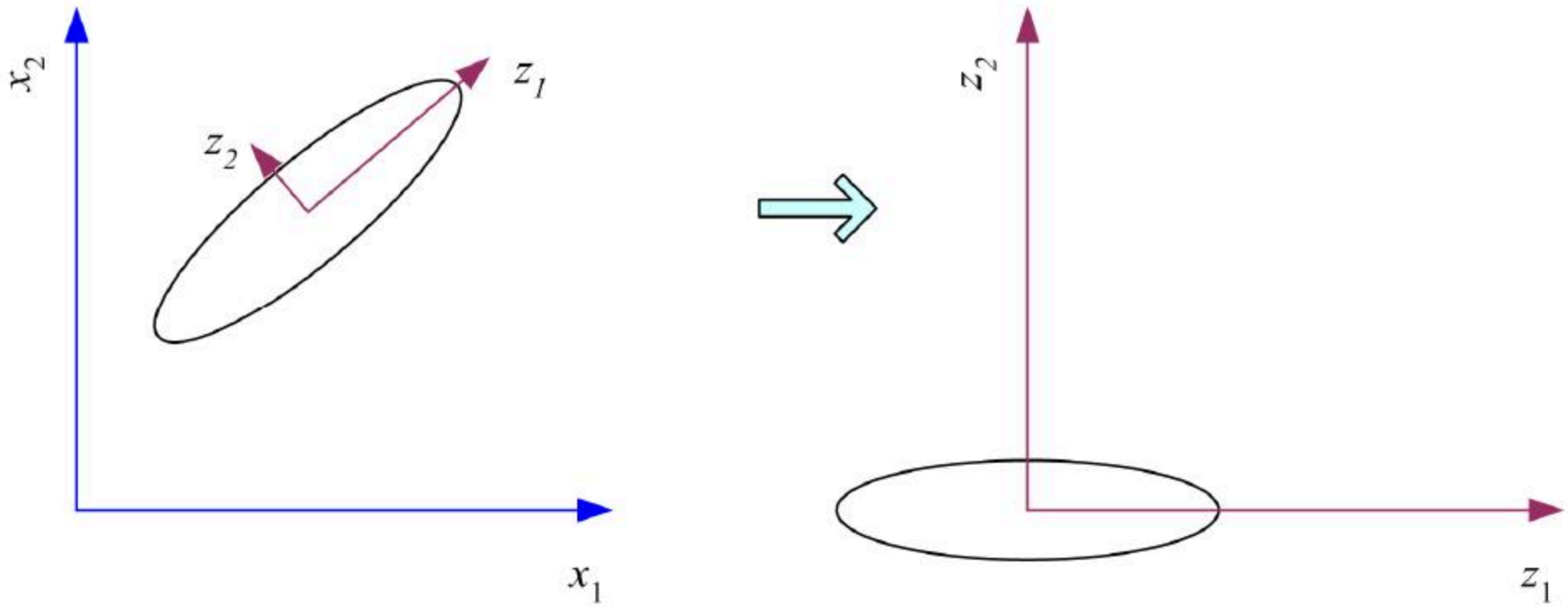
- continue k times*

Linear Feature extraction - PCA

What PCA does

$$z = W^T(x - m)$$

where $W_{.,j}$ is the j^{th} eigenvector of Σ, and m is the sample mean.

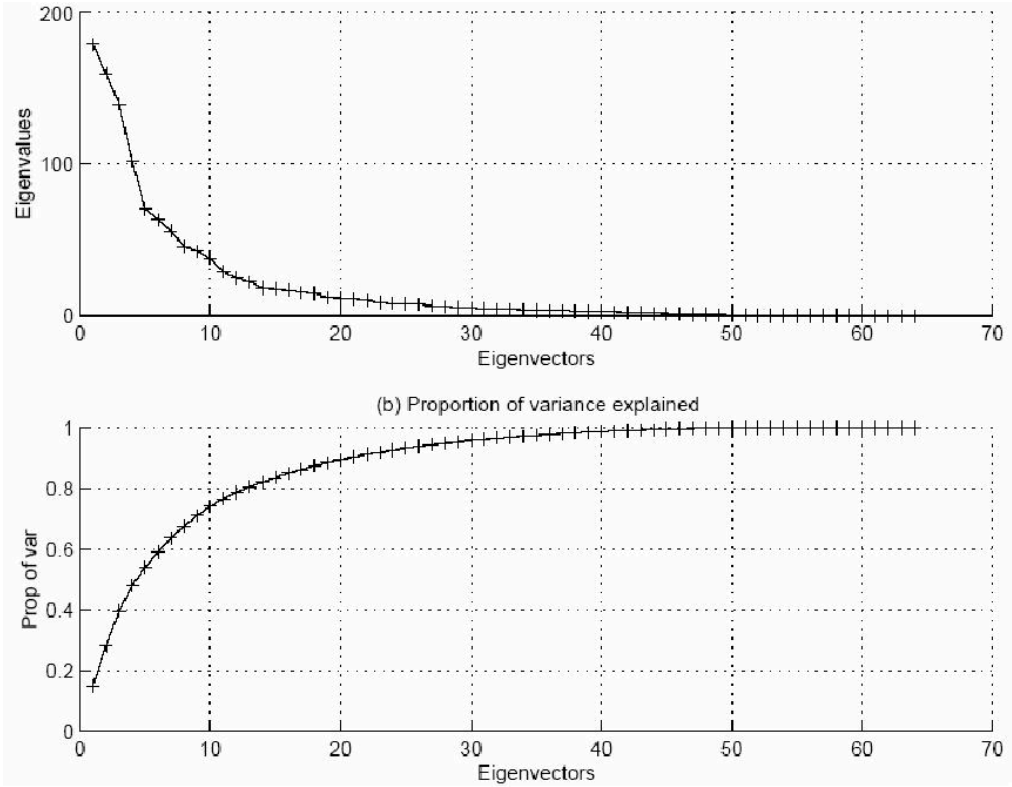


Linear Feature extraction - PCA

POV: proportion of variance.

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

- Typically, k/ POV$>$threshold
- Scree graph plot of POV, stop at elbow



Linear Feature extraction - PCA

```
from sklearn.decomposition import PCA
X = ...
pca = PCA(n_components=2)
pca.fit(X)
PCA(n_components=2)
print(pca.explained_variance_ratio_)
```

Linear Feature extraction - MDS

Given pairwise distances between N points,$d_{ij}, i, j \in \{1 \cdots N\}$, place on a low dimensional map such as distances are preserved.

Sammon stress

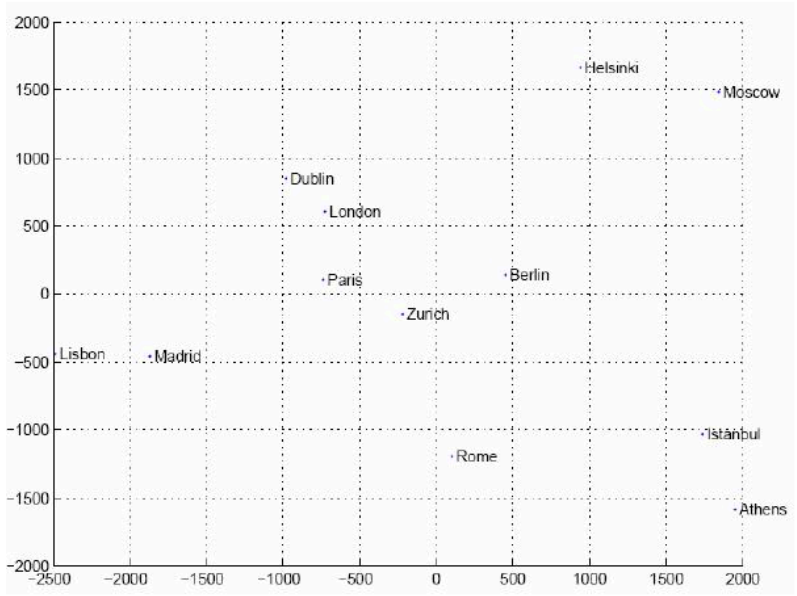
$$z = g(x|\theta)$$

Find θ minimizing

$$\mathbb{E}(\theta|X) = \sum_{r,s} \frac{\left(\|z^r - z^s\| - \|x^r - x^s\|\right)^2}{\|x^r - x^s\|^2}$$

Feature extraction - MDS

```
from sklearn.manifold import MDS
X = ..
embedding = MDS(n_components=2)
X2 = embedding.fit_transform(X)
```



Linear Feature extraction - MDS

Example: 3D points of swiss roll given by their distance matrix.

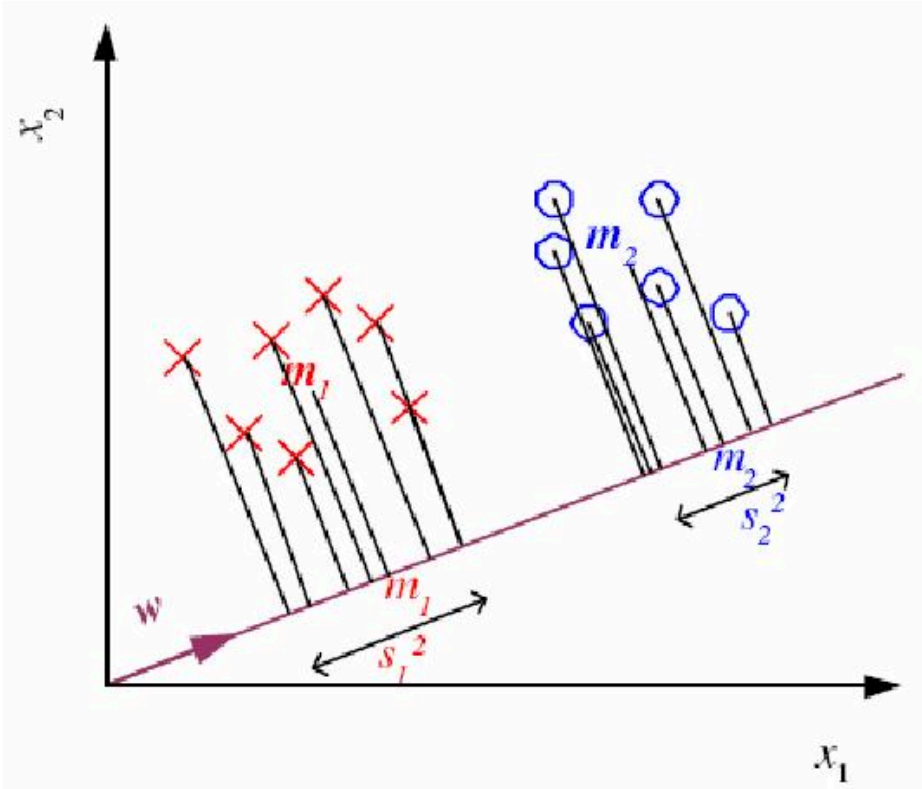
Linear Feature extraction - LDA

Linear (or Fisher) Discriminant Analysis

Find a low-dimensional space such that when x is projected, classes are well-separated.

$$\max_{w} \frac{m_1 - m_2}{s_1^2 + s_2^2}$$

- $m_i = \dfrac{\sum_t w^T x_t r_t}{\sum_t r_t}$
- $s_i = \sum_t r_t \left(w^T x_t - m_i\right)^2$



Linear Feature extraction - LDA

Data Scattering

$$\max_{w} \frac{m_1 - m_2}{s_1^2 + s_2^2}$$

Inter class

$$\begin{aligned}(m_1 - m_2)^2 &= \left(w^T\mu_1 - w^T\mu_2\right)^2 \\ &= w^T(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w = w^T S_B w\end{aligned}$$

where $S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$

Linear Feature extraction - LDA

Data Scattering

$$\max_{w} \frac{m_1 - m_2}{s_1^2 + s_2^2}$$

Intra class

$$\begin{aligned} s_1^2 &= \sum_t r_t \left(w^T x_t - m_1\right)^2 \\ &= \sum_t r_t w^T (x_t - m_1)(x_t - m_1)^T w = w^T S_1 w \end{aligned}$$

where $S_1 = \sum_t r_t (x_t - m_1)(x_t - m_1)^T$

$s_1^2 + s_2^2 = w^T S_W w, S_W = S_1 + S_2$

Linear Feature extraction - LDA

Fisher's discriminant

Find w maximizing

$$\frac{w^T S_B w}{w^T S_W w}$$

Solutions

- LDA: $w = c.\, S_W^{-1}(m_1 - m_2)$
- Parametric: $w = \Sigma^{-1}(\mu_1 - \mu_2)$, when $p(x|C_i) \approx \mathcal{N}(\mu_i, \Sigma)$

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
X,y = ..
lda = LinearDiscriminantAnalysis(n_components=2)
X2 = lda.fit(X, y).transform(X)
```

Linear Feature extraction - LDA

Fisher's discriminant

What about the multiple class case ($C > 2$) ?

- Inter class: $S_B = \sum_{i=1}^{C} N_i(\mu_i - \mu)(\mu_i - \mu)^T \quad m = \frac{1}{C}\sum_{i=1}^{C} \mu_i$
- Intra class: $S_W = \sum_{i=1}^{C} S_i = \sum_{i=1}^{C} r_{t,i}\,(x_t - m_i)(x_t - m_i)^T$

$$\max_w \frac{W^T S_B W}{W^T S_W W}$$

$\rightarrow$ The largest eigenvectors of $S_W^{-1} S_B$

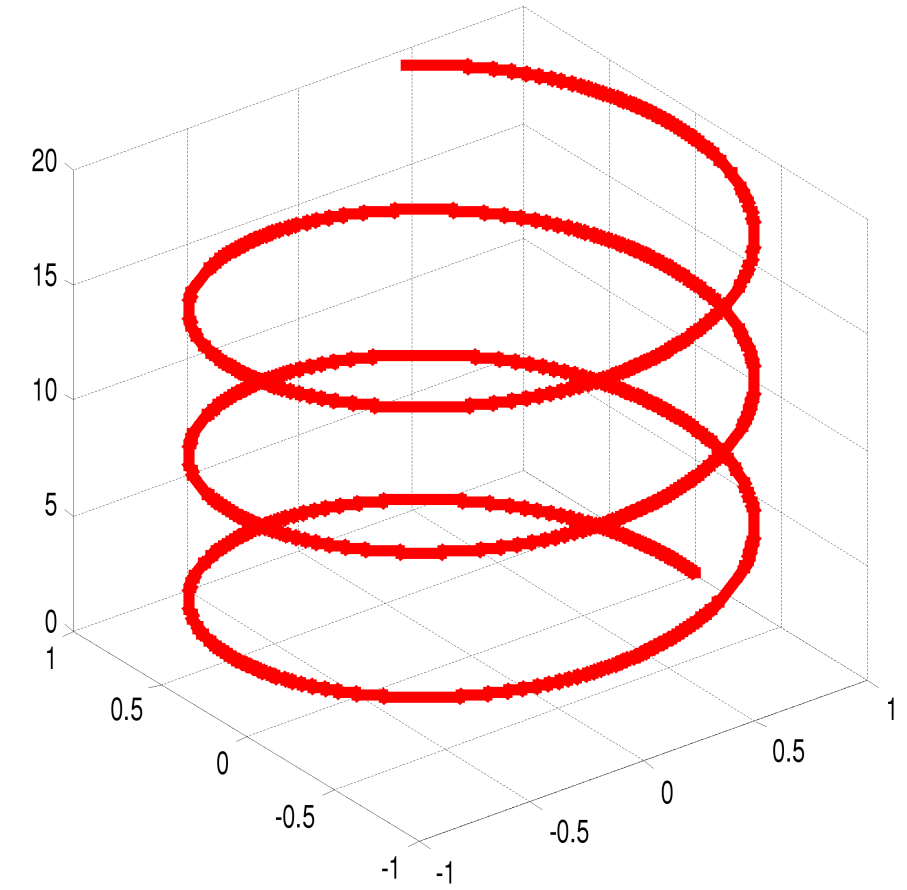
$\rightarrow$ Maximum rank of $C - 1$

Non Linear Feature extraction - Introduction

Deficiencies of Linear Methods

Data may not be best summarized by linear combination of features.

Example: PCA cannot discover 1D structure of a helix



Non Linear Feature extraction - ISOMAP

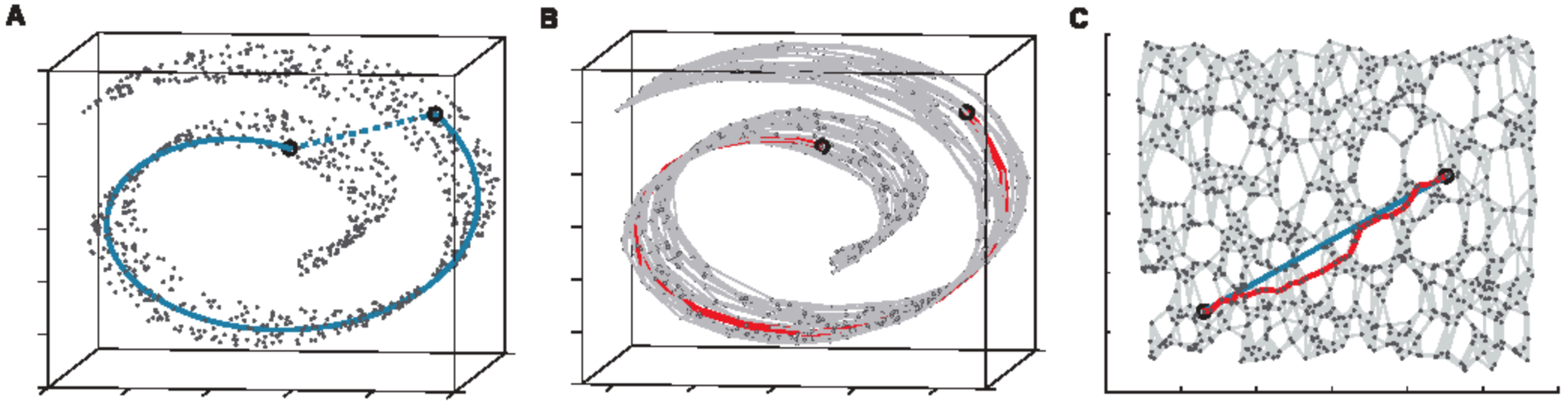
Seminal Paper: J. B. Tenenbaum, V. de Silva and J. C. Langford, A Global Geometric Framework for Nonlinear Dimensionality Reduction, Science 290 (5500) 2000

Algorithm

1. Constructing neighbourhood graph G
2. $\forall$ pair of points in G: shortest path distances $\approx$ geodesic distances.
3. Use MDS with geodesic distances.

Non Linear Feature extraction - ISOMAP

- Construction of the neighbourhood graph G (K-nearest neighborhood (K=7)). D_G: 1000×1000 (Euclidean) distance matrix (fig A)
- Shortest paths in G: D_G:1000×1000 geodesic distance matrix of two arbitrary points along the manifold (fig B)
- Embedding G in $\mathbb{R}^d$ using MDS: Find a d-D Euclidean space preserving pairwise distances (fig C)

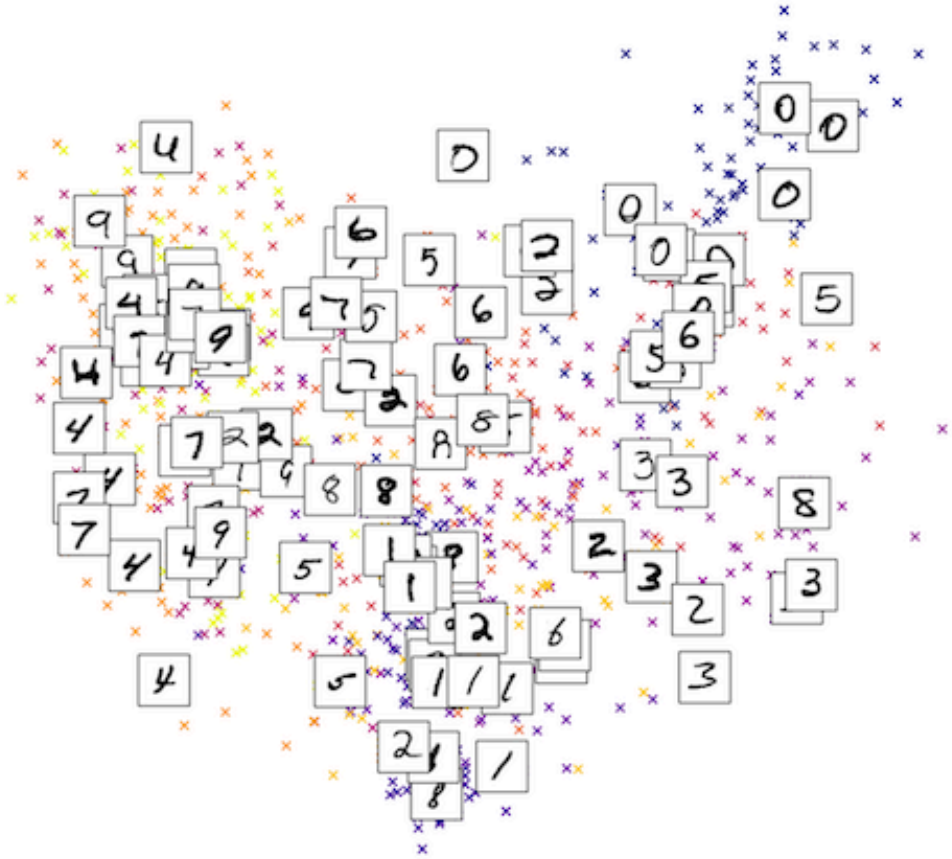


Non Linear Feature extraction - ISOMAP

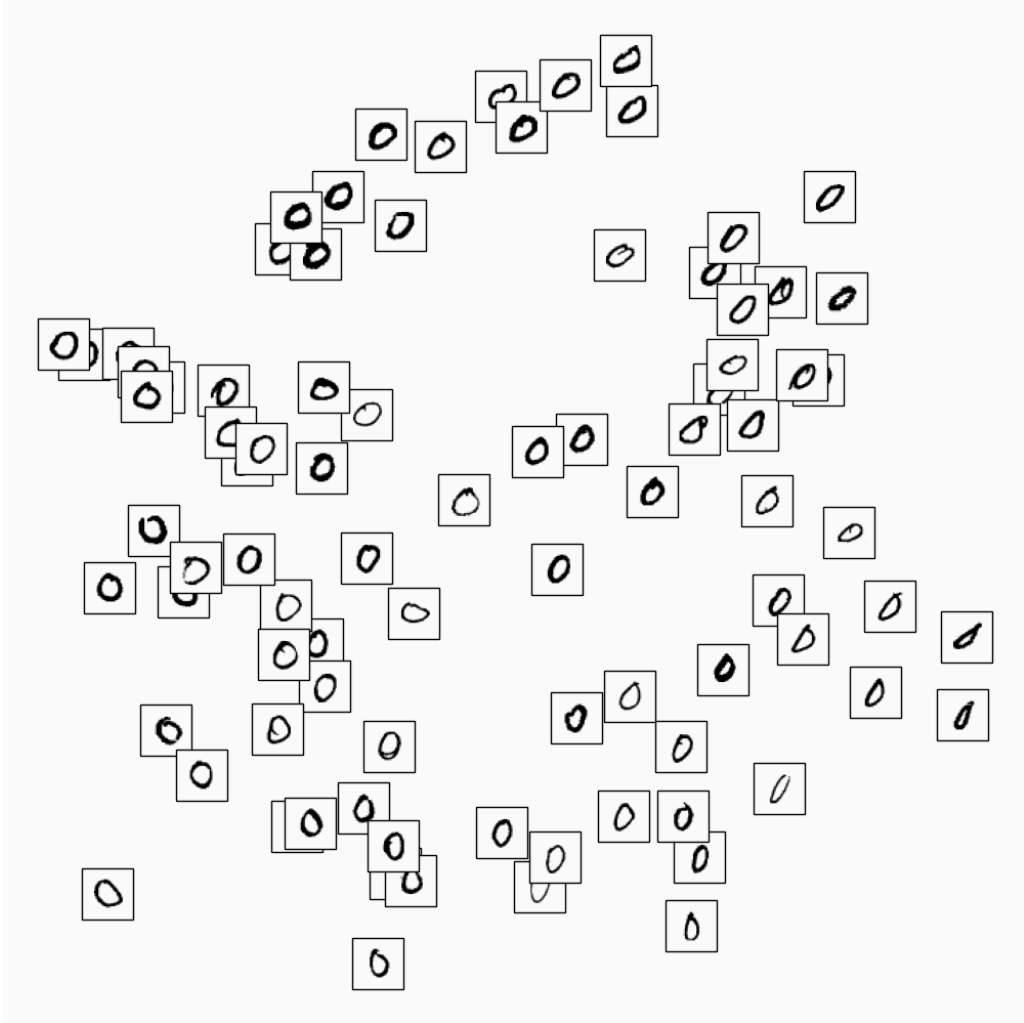
Example: unfolding the swiss roll

Non Linear Feature extraction - ISOMAP

Example: Embedding MNIST data ($\mathbb{R}^{784}$) into $\mathbb{R}^2$



Non Linear Feature extraction - ISOMAP



Non Linear Feature extraction - ISOMAP

Advantages

- Nonlinear
- Globally optimal low-dimensional Euclidean representation even though input space is highly folded, twisted, or curved.
- Guarantee asymptotically to recover the true dimensionality.

Disadvantages

- May not be stable, depends on the topology of the manifold
- asymptotically recover geometric structure of nonlinear manifolds
  - N high: pairwise distances $\approx$ geodesics, but costly
  - N small: geodesic distances very inaccurate
- Distance matrix is dense $\Rightarrow$ does not scale to large datasets $\rightarrow$ Landmark Isomap

Non Linear Feature extraction - ISOMAP

```
from sklearn.manifold import Isomap
X = ...
iso = Isomap(n_components=2)
X2= iso.fit_transform(X)
```

Non Linear Feature extraction - LLE

Seminal Paper} Sam T. Roweis and Lawrence K. Saul, Nonlinear Dimensionality Reduction by Locally Linear Embedding, Science 22:Vol. 290, 2000

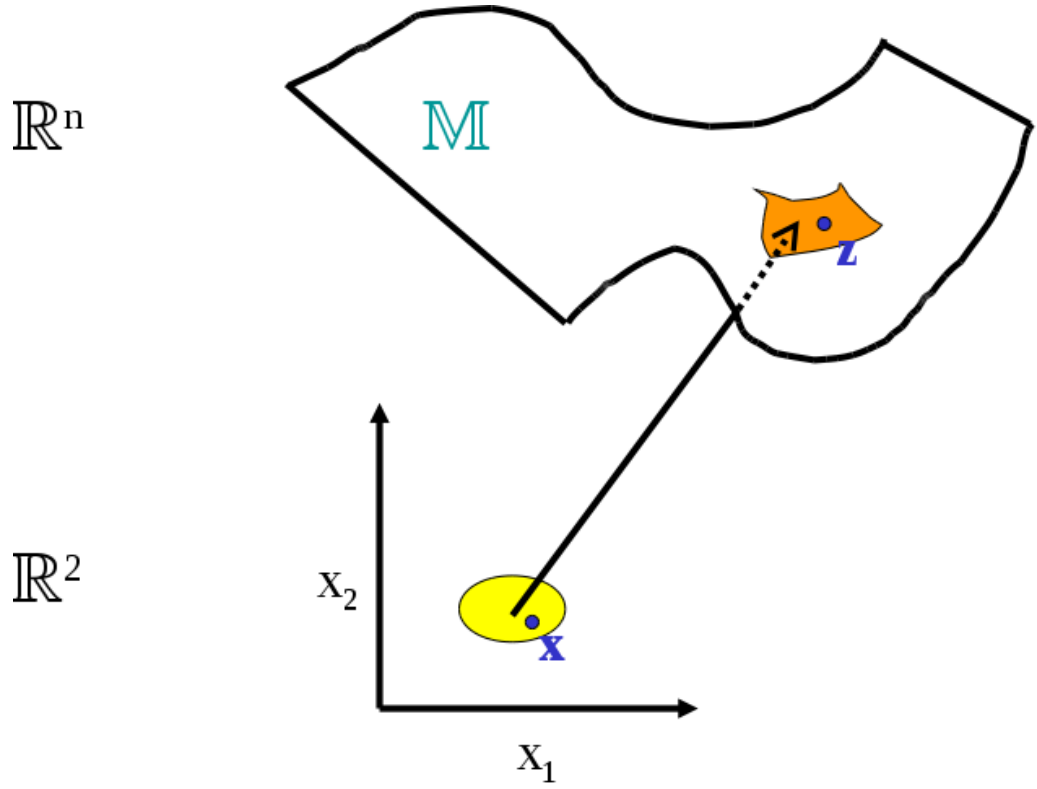
ISOMAP vs. LLE

- Local Linear Embedding $\Rightarrow$ local approach
  $\Rightarrow$ The resulting matrix is sparse...Apply efficient sparse matrix solvers

Non Linear Feature extraction - LLE

Characterictics of a Manifold

- Locally M is a linear patch
- How to combine all local patches together?



Non Linear Feature extraction - LLE

Algorithm

Assumption: manifold $\mathcal{M}$ is roughly linear when viewed locally

Approximation error can be made small:

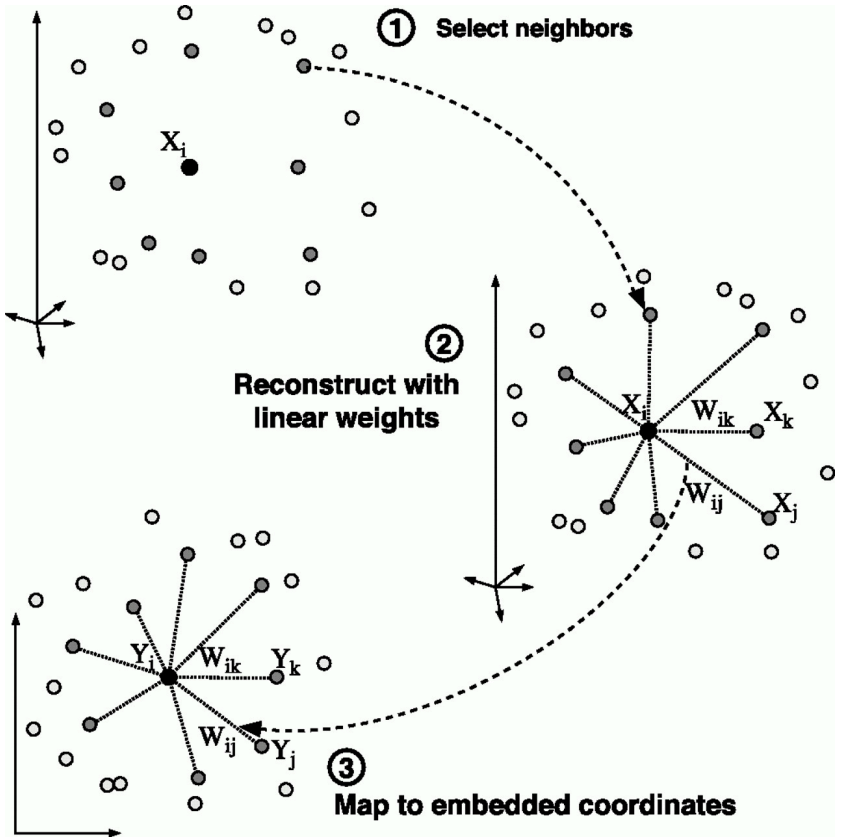
$$Min_W \|x_i - \sum_{j=1}^{k} w_{ij} x_j\|^2$$

1. W: a linear representation of every data point by its neighbors. This is an intrinsic geometrical property of the manifold
2. A good projection should preserve this local geometric property as much as possible

Non Linear Feature extraction - LLE

Algorithm

- We expect each data point and its neighbors to lie on or close to a locally linear patch of $\mathcal{M}$.
- Each point can be written as a linear combination of its neighbors. The weights chosen to minimize the reconstruction error.



Non Linear Feature extraction - LLE

Optimal weights: weights that minimize the reconstruction errors are invariant to rotation, rescaling and translation of the data points.

- Invariance to translation is enforced by adding the constraint that the weights sum to one.
- The weights characterize the intrinsic geometric properties of each neighborhood.

Local geometry is preserved: the same weights that reconstruct the data points in D dimensions should reconstruct it in the manifold in d dimensions.

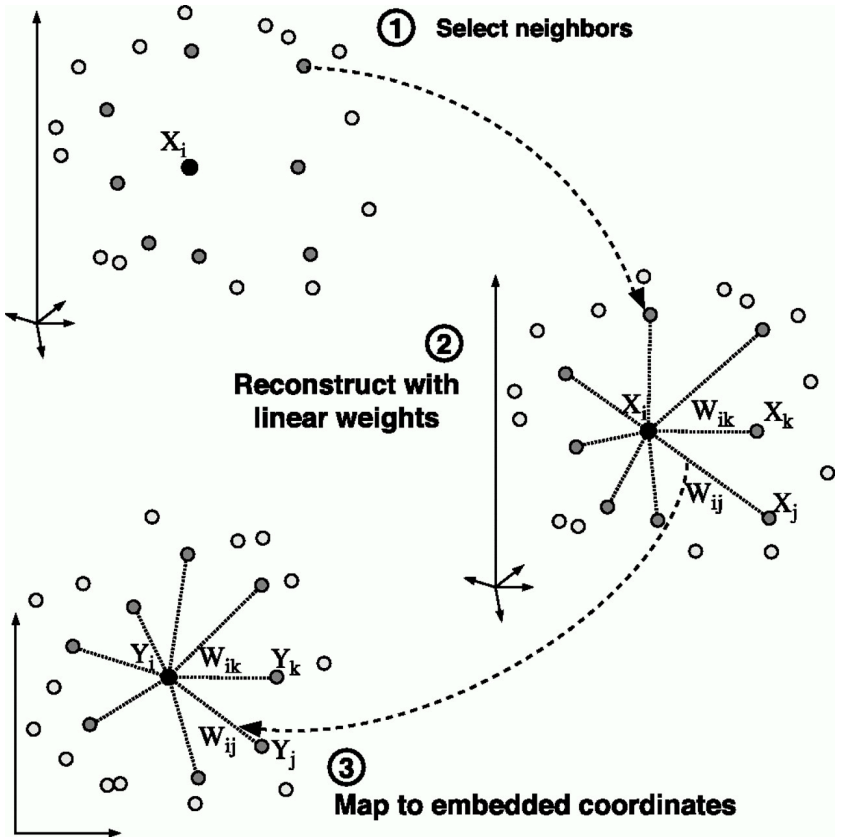
Non Linear Feature extraction - LLE

Algorithm

Low-dimensional embedding
$Y \in \mathcal{M}_{d,N}(\mathbb{R})$

$$\min_{Y} \sum_{i=1}^{N} \|Y_{.,i} - YW_{i,.}\|^2$$

Use the same weights from the original space



Non Linear Feature extraction - LLE

LLE - Constrained LS Problem

Optimization: Compute the optimal weight for each point individually:

$$\|x_i - \sum_{j=1}^{k} w_{ij} x_j\|^2 = \| \sum_{j=1}^{k} w_{ij}(x_i - x_j)\|^2 = \sum_{j=1}^{k} \sum_{k} w_{ij} w_{ik} C_{jk}$$

where $C_{jk} = (x_i - x_j)^T (x_i - x_k)$\
Can be minimized using a Lagrange multiplier for $\sum_j w_{ij} = 1$

Solution:

$$w_{ij} = \frac{\sum_k C_{jk}^{-1}}{\sum_{lm} C_{lm}^{-1}}$$

Non Linear Feature extraction - LLE

LLE space

- $Y_{.,i} \in \mathbb{R}^k$: projected vector for X_i
- The geometrical property is best preserved if $E(Y) = \sum_i \|Y_{.,i} - \sum_j w_{ij} Y_{.,j}\|^2 \quad \text{is small}$
- Y: eigenvectors of the lowest d non-zero eigenvalues of $M = (I - W)^T (I - W)$

$\rightarrow$ Eigenvalue problem}: $E(Y) = Tr(YMY^T)$

$U = (U_1 \cdots U_d)$: bottom eigenvectors of M. Then

$$Y = U^T \text{ and } M_{ij} = \delta_{ij} - w_{ij} - w_{ji} + \sum_k w_{ki} w_{kj}$$

Non Linear Feature extraction - LLE

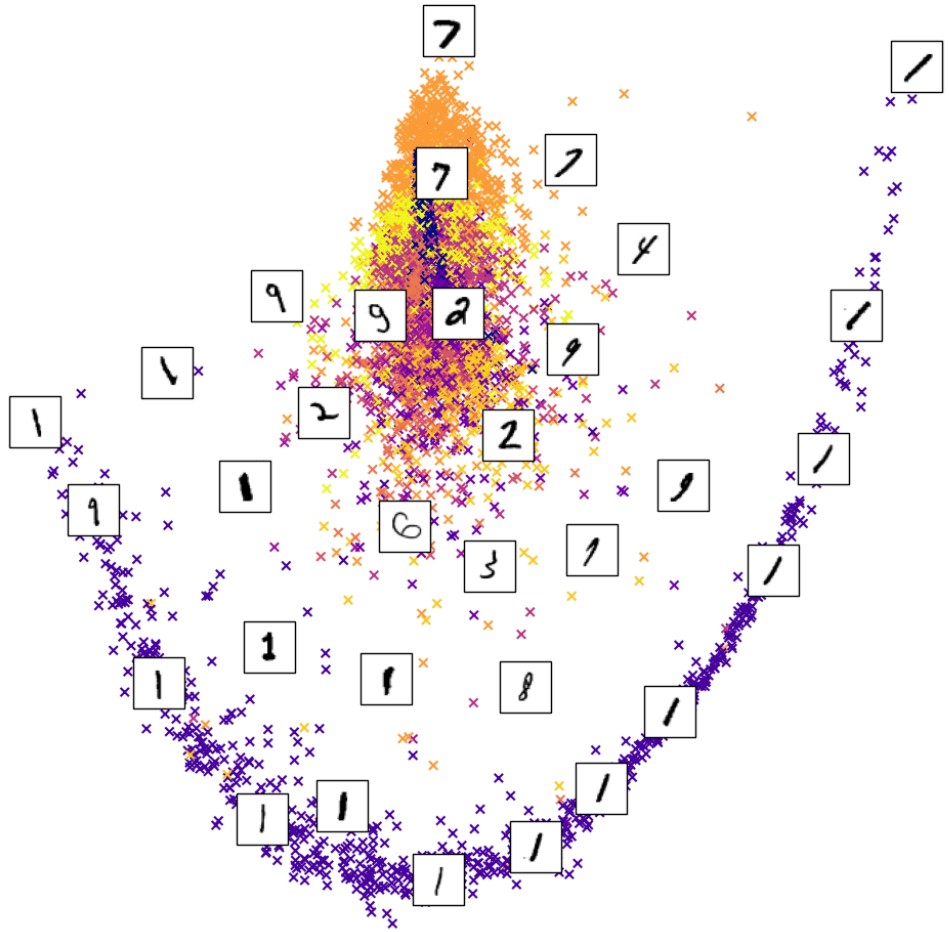
```
from sklearn.manifold import LocallyLinearEmbedding
X = ...
lle = LocallyLinearEmbedding(n_components=2)
X2 = lle.fit_transform(X)
```

Non Linear Feature extraction - LLE

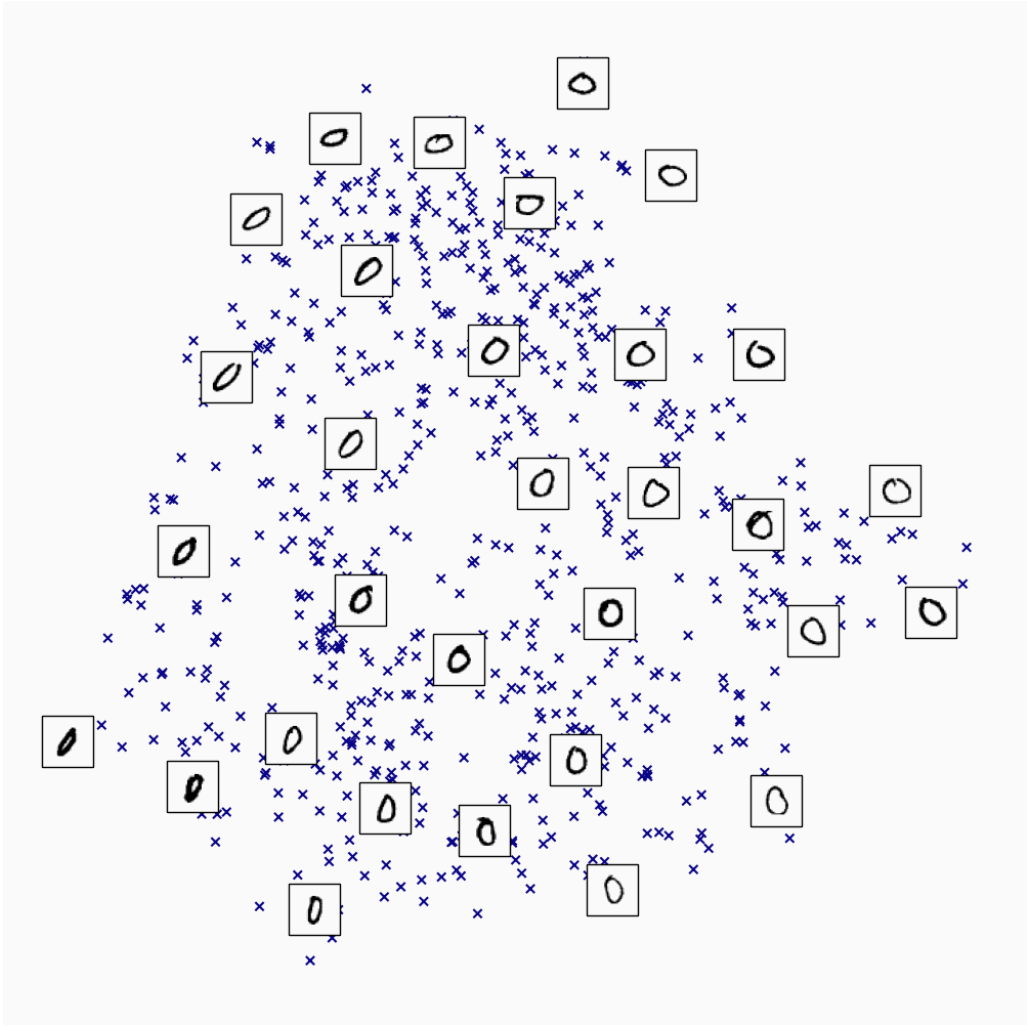
Example: 3D points of swiss roll given by their distance matrix.

Non Linear Feature extraction - LLE

Example: Embedding MNIST data ($\mathbb{R}^{784}$) into $\mathbb{R}^2$



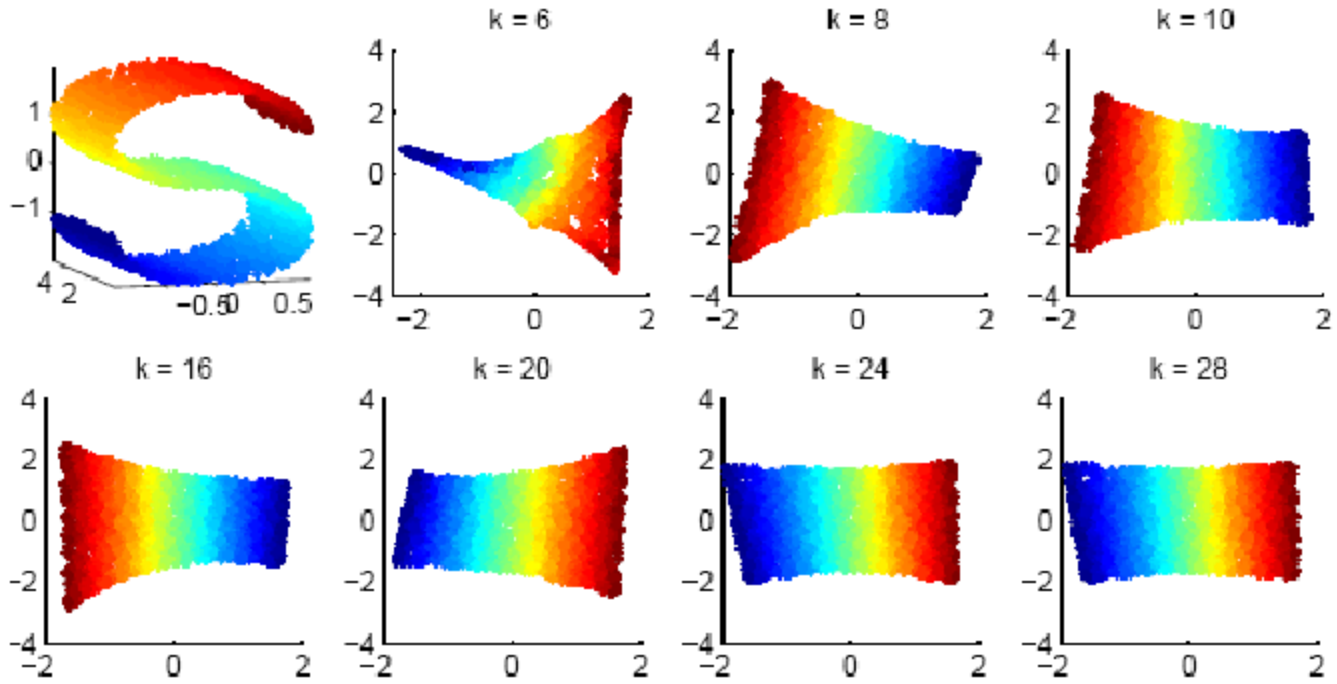
Non Linear Feature extraction - LLE



Non Linear Feature extraction - LLE

Limitations

- Require dense data points on the manifold for good estimation
- Need for a good neighborhood $\Rightarrow$ How to choose k?
  - small $\longrightarrow$ rank deficient tangent space and lead to over-fitting
  - large $\longrightarrow$ Tangent space will not match local geometry well



Non Linear Feature extraction - tSNE

t-Distributed Stochastic Neighbor Embedding: a 4 step algorithm

1. Transform the distance information between data points x_i into conditional probabilities, expressing a similarity relationship. The similarity between x_i and x_j is the conditional probability $P_{j|i}$, which expresses the fact that x_i will consider x_j as its neighbor according to a Gaussian probability density centered at x_i and of variance σ_i.

$$P_{j|i} = \frac{e^{-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}}}$$

Non Linear Feature extraction - tSNE

t-Distributed Stochastic Neighbor Embedding: a 4 step algorithm

2. In the same way, we calculate on $\mathcal{M}$

$$Q_{j|i} = \frac{e^{-\frac{\|y_i - y_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{\|y_i - y_k\|^2}{2\sigma_i^2}}}$$

By minimizing the difference between $P_{j|i}$ and $Q_{j|i}$, the Stochastic Neighbor Embedding (SNE) algorithm attempts to preserve similarities. SNE minimizes the sum of Kullback-Leibler divergences by gradient descent:

$$D_{KL}(P, Q) = \sum_{i \neq j} P_{j|i} log\left(\frac{P_{j|i}}{Q_{j|i}}\right)$$

Non Linear Feature extraction - tSNE

t-Distributed Stochastic Neighbor Embedding: a 4 step algorithm

3. tSNE uses other definitions of conditional probabilities in the original space and $\mathcal{M}$.

$$P_{ij} = \frac{P_{j|i} + P_{i|j}}{2n}$$

with $P_{ii} = 0$ for all i, and

$$Q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_l - y_k\|^2\right)^{-1}}$$

minimization being performed on the Kullback Leibler divergence. As this cost function is non-symmetrical, optimization favors modeling high values of P_{ij} by high values of Q_{ij}.

$$D_{KL}(P, Q) = \sum_{i \neq j} P_{ij} log\left(\frac{P_{ij}}{Q_{ij}}\right)$$

Non Linear Feature extraction - tSNE

t-Distributed Stochastic Neighbor Embedding: a 4 step algorithm

4. the variance depends on the observation point. Each value of σ_i induces a probability distribution P_i over the set of x_i points, whose entropy $H(P_i)$ increases with σ_i.

tSNE performs a binary search for a value of σ_i allowing P_i to reach a fixed perplexity $Perp(P_i) = 2^{H(P_i)}$, with $H(P_i) = -\sum_j P_{ij} log_2 P_{ij}$.

Perplexity can be interpreted as a measure of the average number of effective neighbors.

Non Linear Feature extraction - tSNE

Example: Embedding MNIST data ($\mathbb{R}^{784}$) into $\mathbb{R}^2$