Unsupervised Learning

- The methods we considered so far were focused on Supervised methods such as classification and regression
- In that setting, both a set of predictors (features) $X_1, X_2, \ldots, X_n$ and a response variable Y are available for each data observation
- Now, we will consider Unsupervised methods where an output variable Y is missing

- The goals of the Unsupervised Learning are to discover interesting facts about the observed data. Can we visualize the data in an informative way? Are there subgroups among the variables or among the observations?

- Two general tasks are considered next

  - Dimensionality Reduction – used to visualize the data or for data-preprocessing before supervised techniques are applied

  - Clustering – a broad class of methods for discovering unknown subgroups in the data

Challenges

- Unsupervised learning (UL) is more subjective then SL as there is no simple analysis goal such as prediction of a response
- The techniques for UL are growing in importance, e.g.
  - Find groups of shoppers characterized by their browsing and purchase history
  - Movies grouped by the ratings assigned by movie viewers
- On the other side, it is easier to obtain unlabeled data, e.g. from a lab instrument or a computer, than labeled data which can require human intervention

We will follow the exposition in Chapters 8 & 9 in "Hands-on ML with Scikit-Learn,…" by A. Geron (HOML in short)

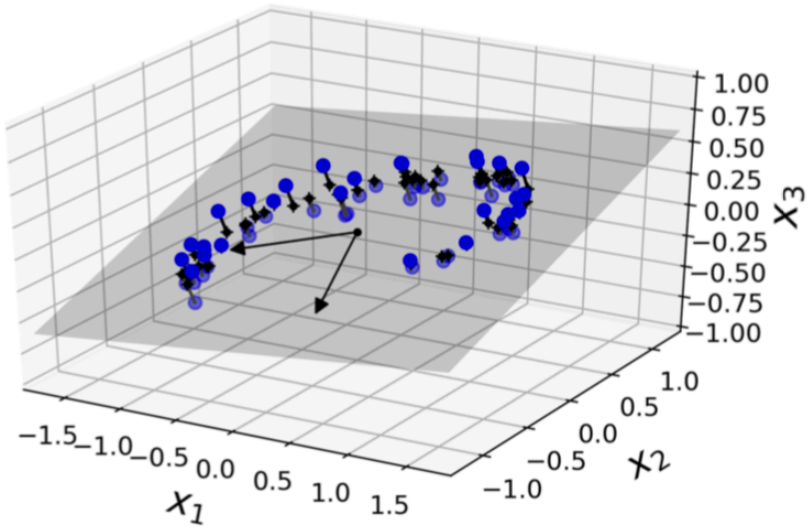
Dimensionality Reduction

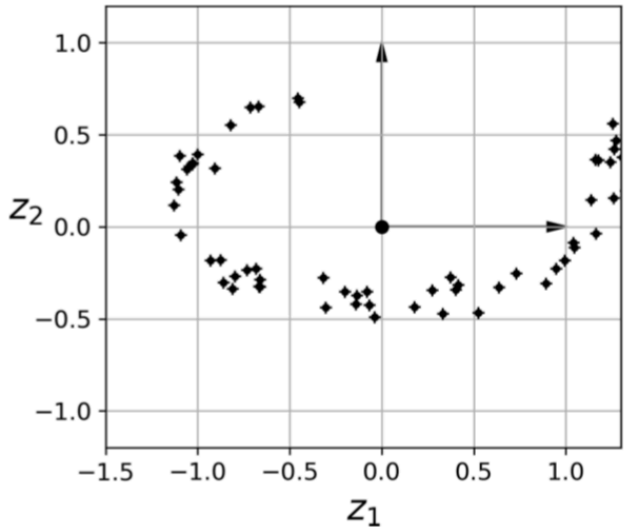
The main approaches are projection and Manifold Learning

Projection

- Data is projected onto a lower dimensional space exploiting the fact that data observations usually lie within or close to a much lower-dimensional subspace of the original data space

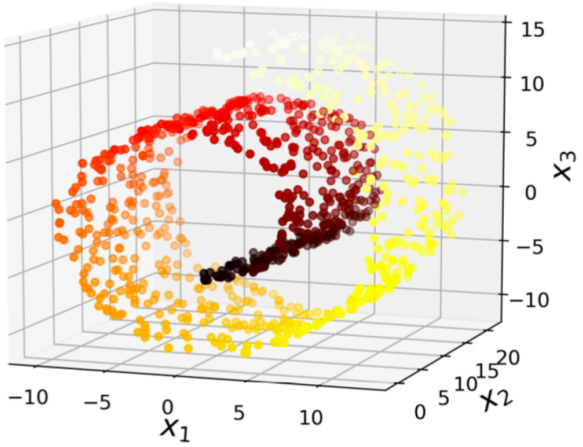
- As an example, let's consider the data on the left plot below.





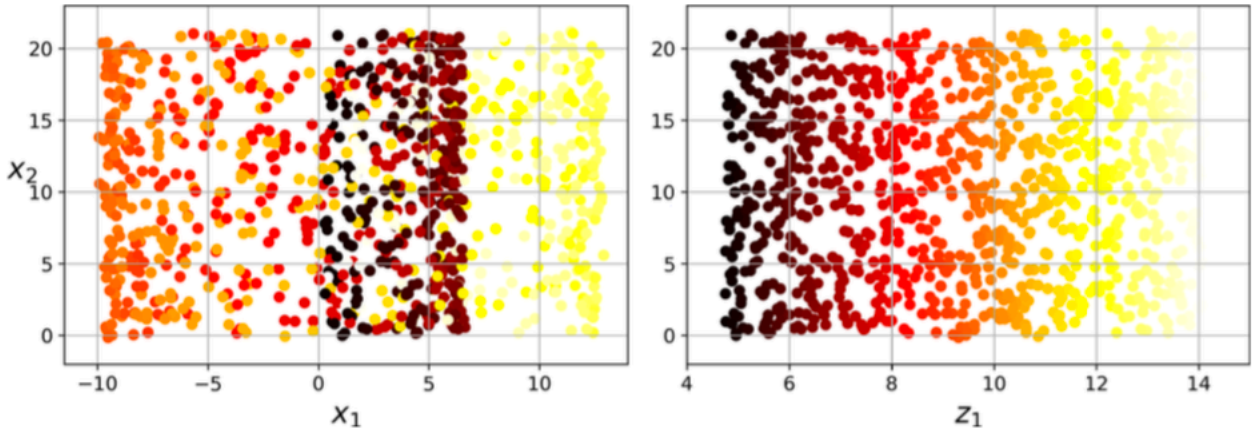
- The 3D dataset (blue dots) lies close to a 2D subspace (the grey plane)
- If we project every training instance perpendicularly onto this subspace (as represented by the short lines connecting the instances to the plane), we get the new 2D dataset shown on the right plot. The new features z_1 and z_2 are the coordinates of the projections on the plane.

- Projection is not always the best approach to dimensionality reduction - consider the famous *Swiss roll* toy dataset



If we simply project onto the $(x_1\ x_2)$ plane, the different layers of the roll are squished together and mixed – see left below.

We really want to unroll the set and get the result on the right.

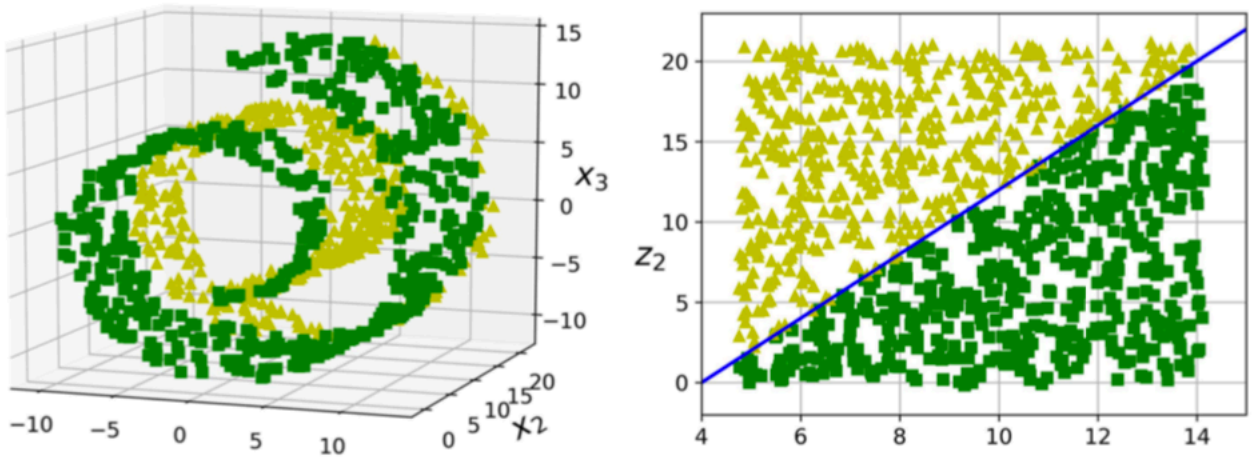


Manifold Learning

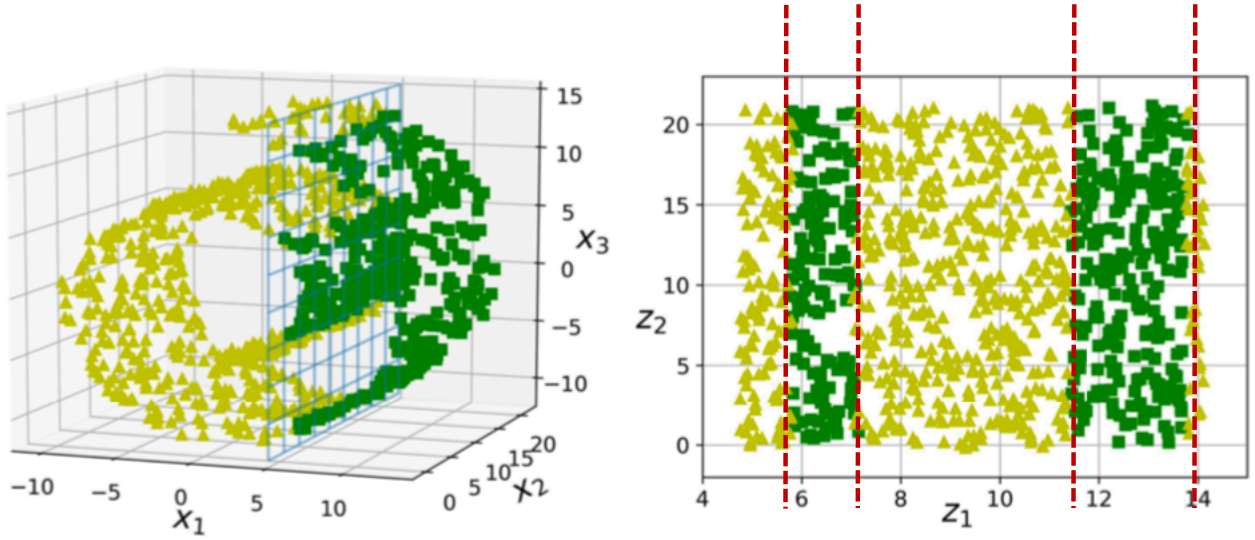
- The Swiss roll is an example of a 2D manifold - it locally resembles a 2D plane, but it is rolled in the third dimension.
- In generally, a d-dimensional manifold is a part of an n-dimensional space (where $d < n$) that locally resembles a d-dimensional hyperplane. In the case of the Swiss roll, $d = 2$ and $n = 3$.
- Many dimensionality reduction algorithms work by modeling the manifold on which the training instances lie; this is called Manifold Learning. It relies on the manifold assumption/ hypothesis, which holds that most real-world high-dimensional datasets lie close to a much lower-dimensional manifold. This assumption is very often empirically observed.
- In addition, it's implicitly assumed that the task at hand (e.g., classification or regression) will be simpler if expressed in the lower-dimensional space of the manifold.

- It's confirmed in the case of the Swiss roll when it's split into 2 classes as on the left plot below.

- The decision boundary in the 2D (unrolled) manifold is just a straight line, while it looks quite complicated in the original space.



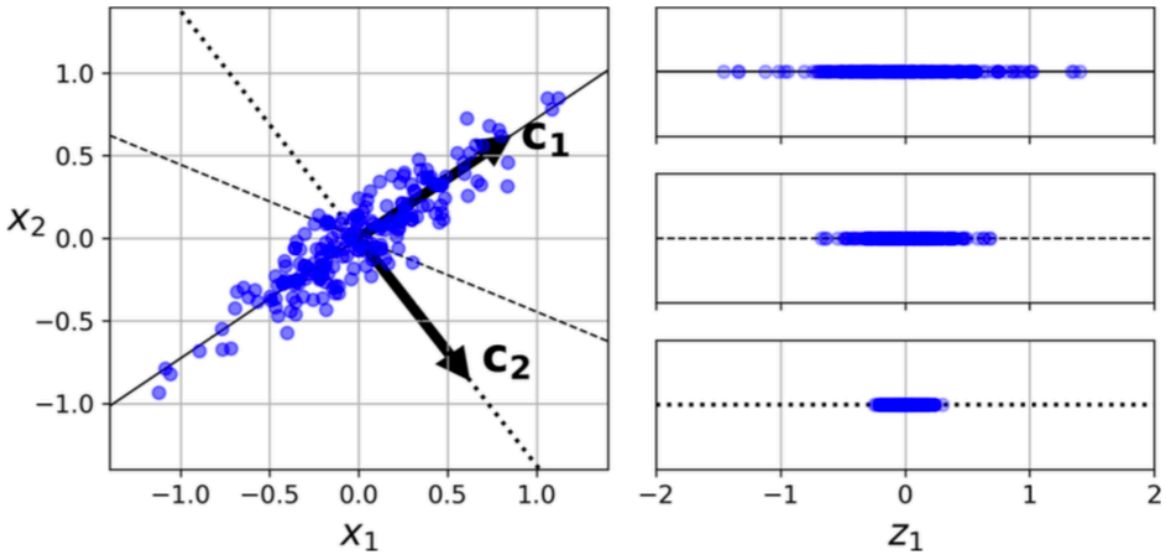
- However, this implicit assumption does not always hold. For example, in the left plot below, the decision boundary in the original 3D space is pretty simple: a vertical plane located at $x_1 = 5$. It is more complex in the unrolled manifold (a collection of 4 independent line segments):



- So, reducing the dimensionality of your training set before training a model will usually speed up training, but it may not always lead to a better or simpler solution; it all depends on the dataset.

Principal Component Analysis (PCA)

- PCA produces low-dimensional representation of a dataset. It finds the hyperplane that lies closest to the data and then projects the data onto it.
- To find the projection hyperplane, the sequence of linear combinations of the variables that have max variance and are mutually uncorrelated is first determined. The new vectors defined by these linear combinations are called the principal components.
- This process is illustrated (in 2 dimensions) with the following plots



- Different choices for the 1st direction/axis and the variance of the projected data (on the right) are shown above. The axis accounting for the largest amount of variance is $\boldsymbol{c_1}$ and it's called the 1st principal component (PC).
- Then a 2nd axis, orthogonal to the first one, that accounts for the largest amount of remaining variance is found. In this 2D example there is no choice: it is the dotted line aligned with $\boldsymbol{c_2}$. If it were a higher-dimensional dataset, PCA would also find a 3 axis, orthogonal to both previous axes and so on – as many axes as the dimension p of the (training) dataset

- The data matrix $\boldsymbol{X}$ is $n \times p$ (n rows corresponding to the number of observations and p columns for the features $X_1, X_2, \ldots, X_p$).

- All PCs could be found using the Singular Value Decomposition (SVD) of the data matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = \boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^{\boldsymbol{T}}$$

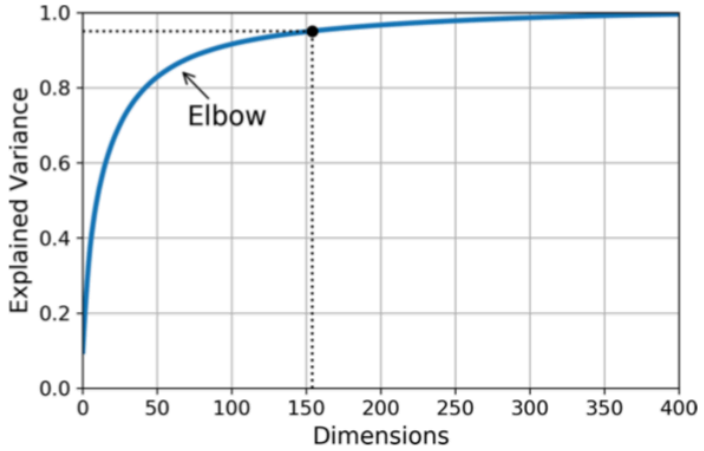
  Matrix $\boldsymbol{V}$ has as its columns the PCs $\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_p}$ with $\|\boldsymbol{c_i}\| = 1, i = 1, \ldots, p$.

- To project the training set onto the hyperplane and obtain a reduced dataset $\boldsymbol{X}_{d-proj}$ of dimensionality d, define the matrix $\boldsymbol{W_d}$ to have as columns the first d principal component direction and compute

$$\boldsymbol{X}_{\text{recovered}} = \boldsymbol{X}_{d-proj}\,\boldsymbol{W_d^T}$$

  Column i is $\boldsymbol{c_i} = \left(w_{1i}\ w_{2i}\ \ldots\ w_{pi}\right)^T$ for $i = 1,2, \ldots, d$. The w_{ji}'s are called the loadings for the i^{th} principal component, the elements of $\boldsymbol{X}_{d-proj}$ - scores.

- Choosing the Right Number of Dimensions
  - We could choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%). Unless, of course, you are reducing dimensionality for data visualization—in that case you will want to reduce the dimensionality down to 2 or 3.
  - Another option is to plot the explained variance as a function of the number of dimensions. There will usually be an elbow in the curve, where the explained variance stops growing fast. In the plot below, you can see that reducing the dimensionality down to about 100 dimensions wouldn't lose too much explained variance.

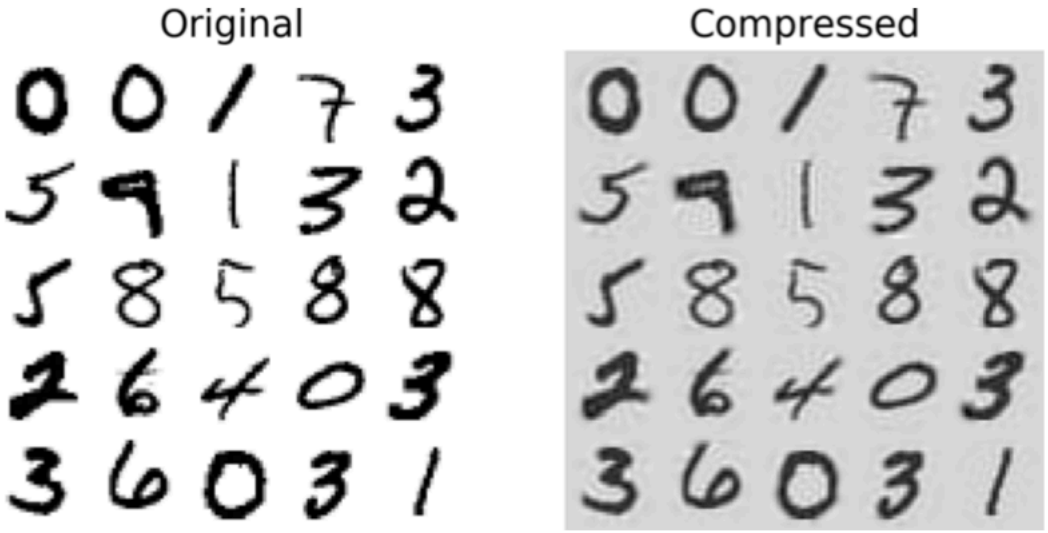


PCA for Compression

- After dimensionality reduction, the training set takes up much less space. E.g. try applying PCA to the MNIST dataset while preserving 95% of its variance. You should find that each instance will have just over 150 features, instead of the original 784 features. So, while most of the variance is preserved, the dataset is now less than 20% of its original size! This is a reasonable compression ratio which can speed up a classification algorithm (such as an SVM classifier) tremendously.

- It is also possible to decompress the reduced dataset back to 784 dimensions by applying the inverse transformation of the PCA projection. This won't give you back the original data, but it will likely be close to the original data. The mean squared distance between the original data and the reconstructed data (compressed and then decompressed) is called the *reconstruction error*.

- Below a few digits from the original training set are shown on the left. On the right are the corresponding digits after compression and decompression. You can see that there is a slight image quality loss, but the digits are still mostly intact.



- The inverse transformation (decompression) is

$$\boldsymbol{X}_{d-proj} = \boldsymbol{X}\,\boldsymbol{W}_{\boldsymbol{d}}$$

Randomized PCA

- Scikit-Learn (Python) can use a stochastic algorithm called Randomized PCA that quickly finds an approximation of the first d principal components – just set svd_solver=”randomized” in function PCA(). This method is much faster than full SVD when $d \ll p$.

- By default, Scikit-Learn automatically uses the randomized PCA algorithm if $n > 500$ or $p > 500$ and d is less than 80% of n or p, or else it uses the full SVD approach. This behavior is controlled by svd_solver.

Incremental PCA

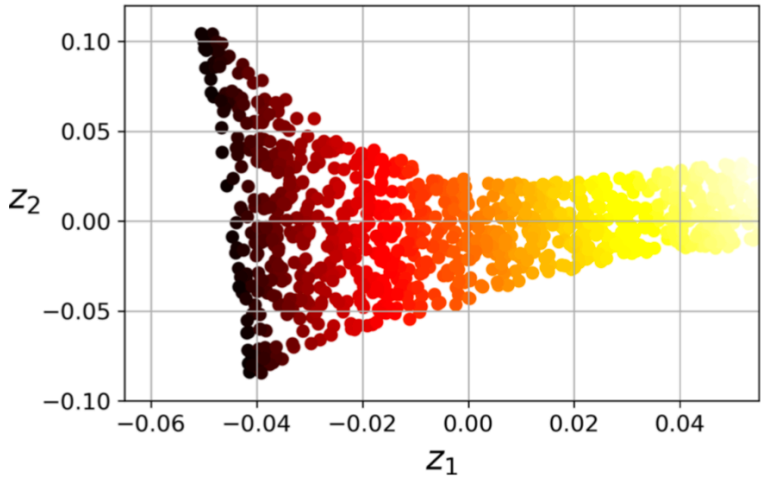
- One problem with the preceding implementations of PCA is that they require the whole training set to fit in memory in order for the algorithm to run.
- Fortunately, Incremental PCA (IPCA) algorithms have been developed. They allow you to split the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time.

- This is useful for large training sets and for applying PCA online (i.e., on the fly, as new instances arrive).

- There is code in the Python notebook which splits the MNIST dataset into 100 mini-batches and feeds them to Scikit-Learn's IncrementalPCA class ffto reduce the dimensionality of the MNIST dataset down to 154 dimensions (just like before).

Locally Linear Embedding (LLE)

- LLE is a Manifold Learning technique that does not rely on projections. In a nutshell, LLE works by first measuring how each training instance linearly relates to its closest neighbors (c.n.), and then looking for a low-dimensional representation of the training set where these local relationships are best preserved (see below).
- LLE is very good at unrolling twisted manifolds, especially when there is not too much noise. For example, it unrolls completely the Swiss roll as can be seen in the LLE result plot below



- The distances between observations are locally well preserved. They are not preserved on a larger scale: the left part of the unrolled Swiss roll is stretched, while the right part is squeezed. Still LLE did a pretty good job at modeling the manifold.

- LLE algorithm: For each instance $\boldsymbol{x}^{(i)}$, the k nearest neighbors are identified and a linear function $\sum_{j=1}^{n} w_{i,j}\boldsymbol{x}^{(j)}$, where $w_{i,j} = 0$ if $\boldsymbol{x}^{(j)}$ is not one of the k closest, is determined such that

  Step1

  $$\widehat{W} = argmin_{\,W} \sum_{i=1}^{n}\left(\boldsymbol{x}^{(i)} - \sum_{j=1}^{n} w_{i,j}\,\boldsymbol{x}^{(j)}\right)^2$$

  subject to $\begin{cases} w_{i,j} = 0 \text{ if } \boldsymbol{x}^{(j)} \text{ is not one of the kNN of } \boldsymbol{x}^{(i)} \\ \sum_{j=1}^{n} w_{i,j}\,\boldsymbol{x}^{(j)} = 1 \; for \; i = 1,2,..,n \end{cases}$

Step 2

Map the training instances $\boldsymbol{x}^{(j)}$ to $\boldsymbol{z}^{(j)}$ which belong to a d-dimensional space ($d < p$) while trying to minimize the distance between them

$$\hat{Z} = argmin_{\,Z} \sum_{i=1}^{n} \left(\boldsymbol{z}^{(i)} - \sum_{j=1}^{n} \widehat{w}_{i,j}\, \boldsymbol{z}^{(j)} \right)^2$$

In the above minimization the weights $\widehat{w}_{i,j}$ are the ones found in Step1.

- LLE can be vary useful but scales poorly to very large data

Other Dimensionality Reduction Tehniques

- t –Distributed Stochastic Neighbor Embedding (t-SNE)
  Reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space (e.g., to visualize the MNIST images in 2D).
- Multidimensional Scaling (MDS)
  Reduces dimensionality while trying to preserve the distances between the instances.
- Linear Discriminant Analysis (LDA)
  Is a classification algorithm, but during training it learns the most discriminative axes between the classes, and these axes can then be used to define a hyperplane onto which to project the data.
  The benefit of this approach is that the projection will keep classes as far apart as possible, so LDA is a good technique to reduce dimensionality before running another classification algorithm such as an SVM classifier.