Dimensionality Reduction

Na Yin, Jue Wang

Curse of Dimensionality

Various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience

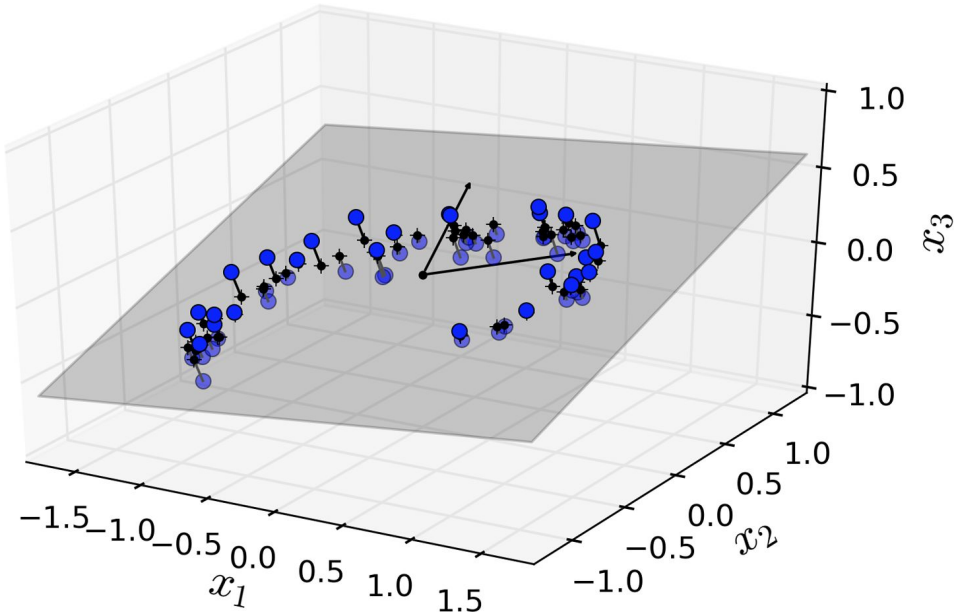
N dimension	Average distance
2	0.52
3	0.66
...	...
1,000,000	408.25

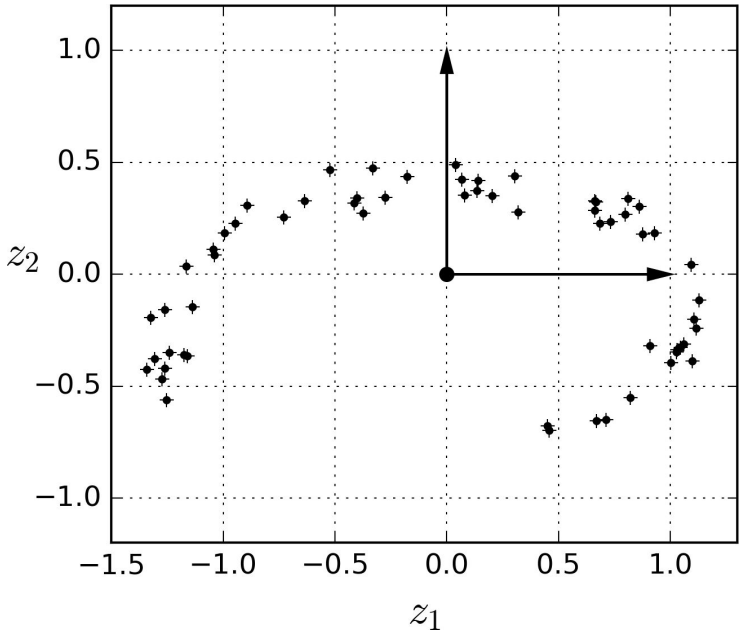
Main Approaches for Dimensionality Reduction

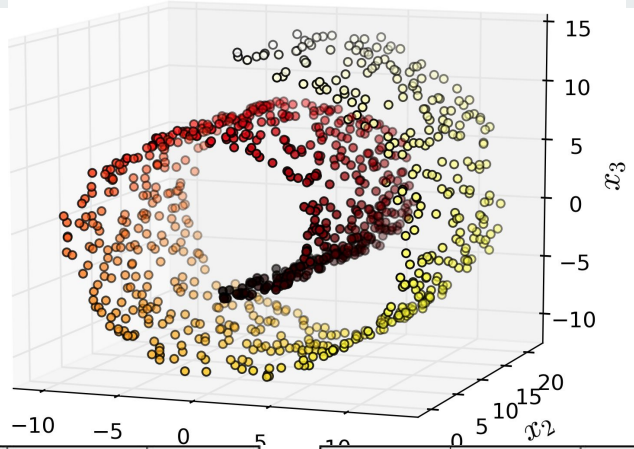
- Projection

- Manifold Learning

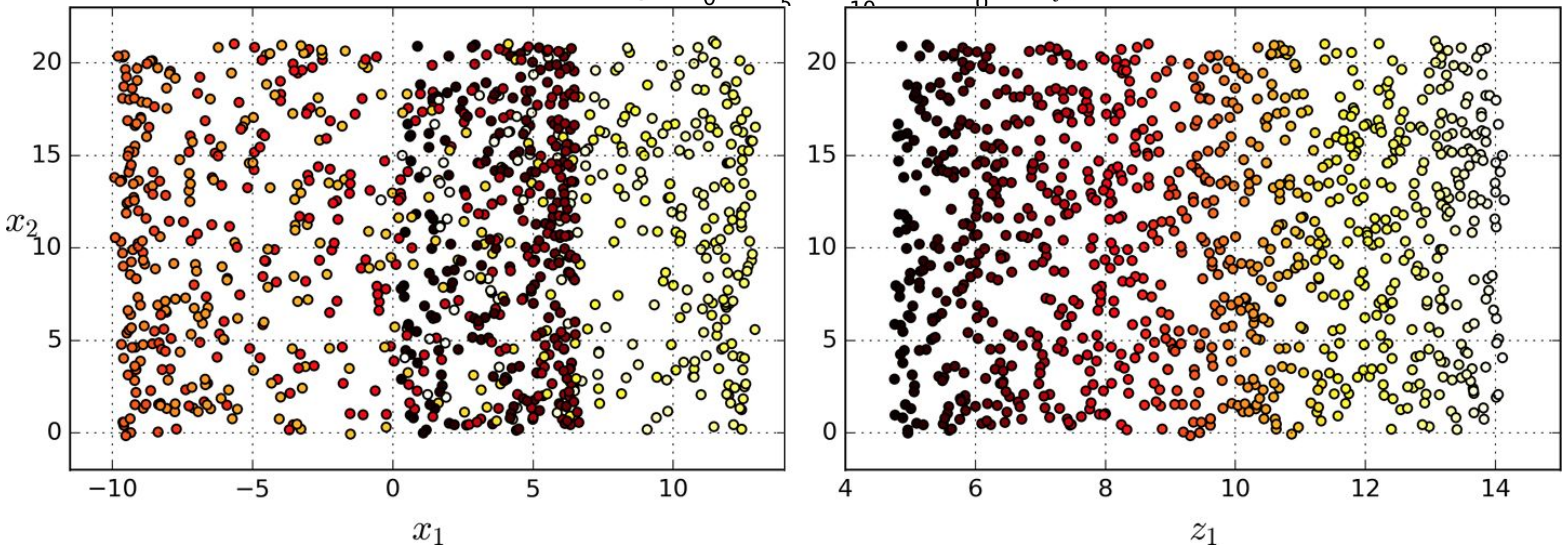
Dimensionality Reduction -- Projection





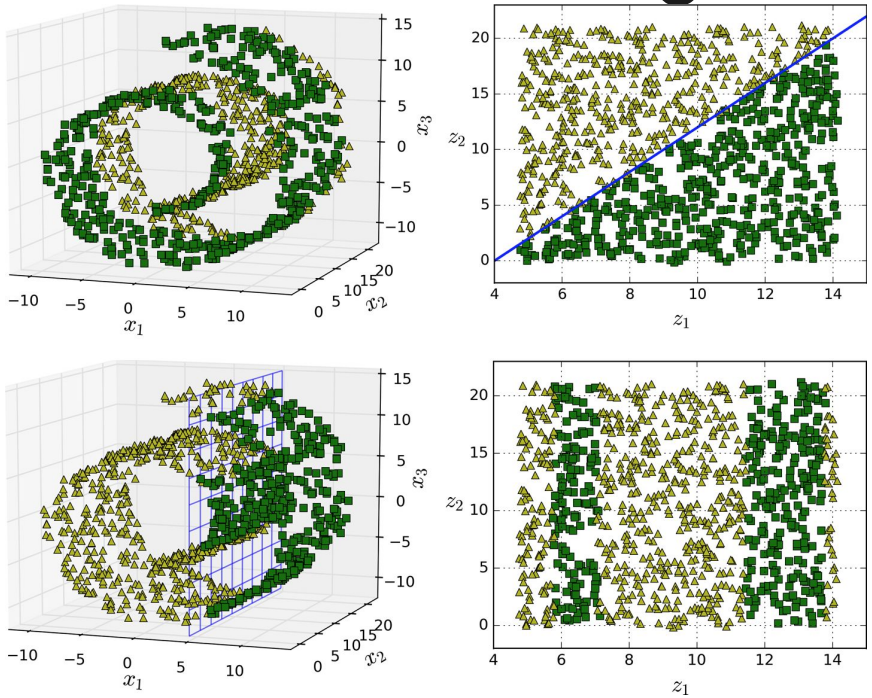


n-dimensional manifold: is a n-dimensional(D) shape that can be bent and twisted in a higher-D space



Dimensionality Reduction -- Manifold Learning

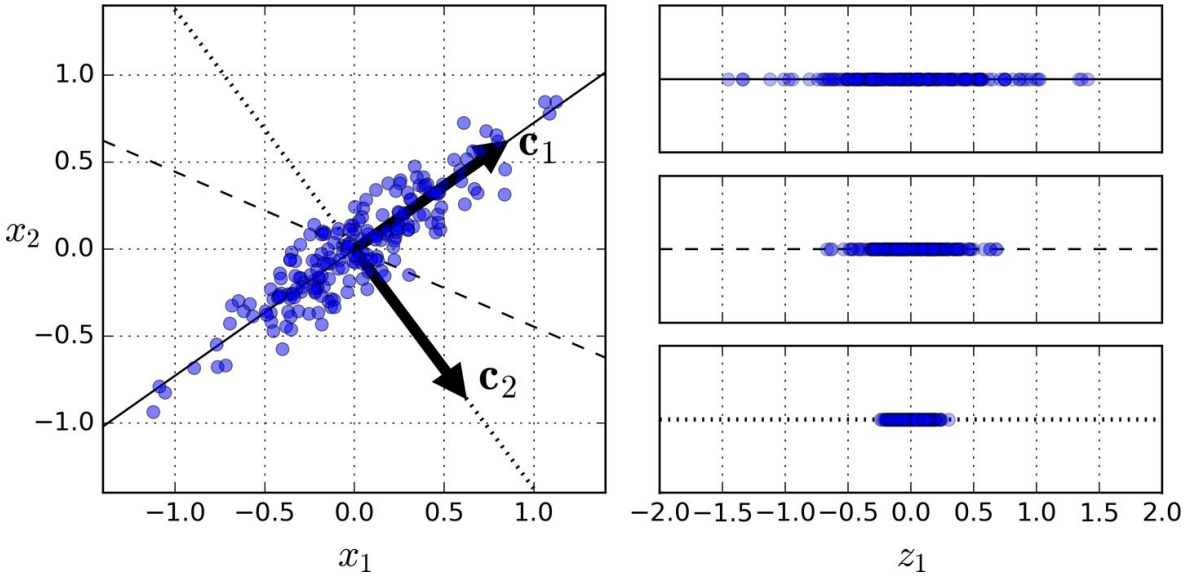
Manifold Learning: find a low-D shape for describing high-D data



PCA - Principal Component Analysis

- PCA is by far the most popular dimensionality reduction algorithm.
  - First identifies the hyperplane that lies closest to the data
  - Then projects the data onto it.

PCA - Preserving the Variance



Results:

- The projection onto the solid line preserve the maximum variance
- The projection onto the dotted line preserve very little variance
- The project onto the dashed line preserves an intermediate amount of variance.

So, it is reasonable to select the axis that preserves the maximum amount of variance, as it will most likely lose less information than the other projections.

PCA - Principal Component

- PCA identifies the axis that accounts for the largest amount of variance in the training set. The unit vector that define the ith axis is called the ith principal component (PC).
- How to find the principal components of the training set?
    - Standard matrix factorization technique - Singular Value Decomposition (SVD): decompose the training set matrix X into the dot product of three matrices U. Σ . V^T, where V, contains all the principal components that we are looking for:
        - X is the real m x n matrix that we wish to decompose
        - U is an m x m matrix
        - Sigma is an m x n diagonal matrix
        - V^T is the transpose of an n x n matrix where T is a superscript.
- PCA assumes that the dataset is centered around the origin.

Equation 8-1. Principal components matrix

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

PCA - Projecting Down to d Dimensions

- Once identified all the principal components, then can reduce the dimensionality of the dataset down to d dimensions by projecting it onto the hyperplane defined by the first d principal component.
- To project the training set onto the hyperplane. Simply compute the dot product of the training set matrix X by the matrix Wd (the matrix composed of the first d columns of V)

```
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X} \cdot \mathbf{W}_d$$

PCA - Explained Variance Ratio

- It indicates the proportion of the dataset's variance that lies along the axis of each principal component.
    - For example, on projection example, 84.2% of the dataset's variance lies along the first axis, and 14.6% lies along the second axis. And there is only 1.2% for the third axis so we can assume that it probably carries little information and we can drop it.
- The demo is the code applies PCA to reduce the dimensionality of the dataset down to 2 dimensions.

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

```
>>> pca.explained_variance_ratio_
array([ 0.84248607,  0.14631839])
```

PCA - Choosing the Right Number of Dimensions

- Instead of arbitrarily choosing the number of dimensions to reduce down to, it is preferable to choose the number of dimensions that add up to a sufficiently large portion of the variance.
- The elbow is the intrinsic dimensionality of the dataset. So reducing the dimensionality down to about 100 wouldn't lose too much explained variance.

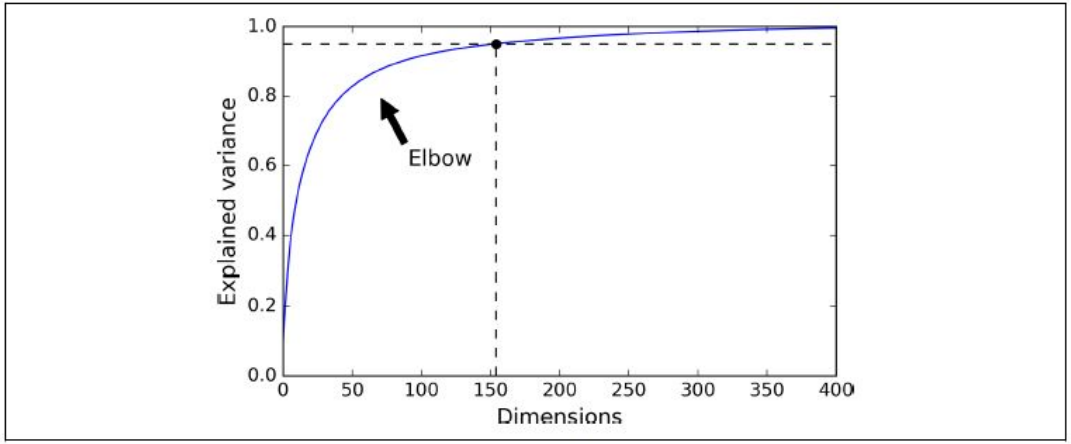


Figure 8-8. Explained variance as a function of the number of dimensions

PCA for Compression

- After taking dimensionality reduction, the training set takes up much less space.
  - MNIST handwritten dataset still preserving 95% of its variance while each instance will have only 150 features instead of the original 784 features.
- It is also possible to decompress the reduced dataset back to original dimensions by applying the inverse transformation. However, this won't give back the original data since the projection lost a bit of information (it will likely be quite close to the original data).
  - The mean squared distance between the original and reconstructed data (compressed and then decompressed) is called reconstruction error.

```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

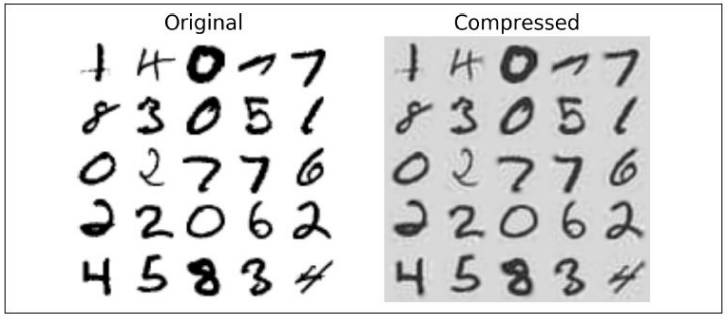


Figure 8-9. MNIST compression preserving 95% of the variance

Incremental PCA

- A replacement for PCA when the dataset to be decomposed is too large to fit in memory
- Split the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time
- Useful for large training sets, and also to apply PCA online

```
n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
    inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

Randomized PCA

Linear dimensionality reduction using approximated Singular Value Decomposition of the data and keeping only the most significant singular vectors to project the data to a lower dimensional space

```
rnd_pca = PCA(n_components=154, svd_solver="randomized")
X_reduced = rnd_pca.fit_transform(X_train)
```

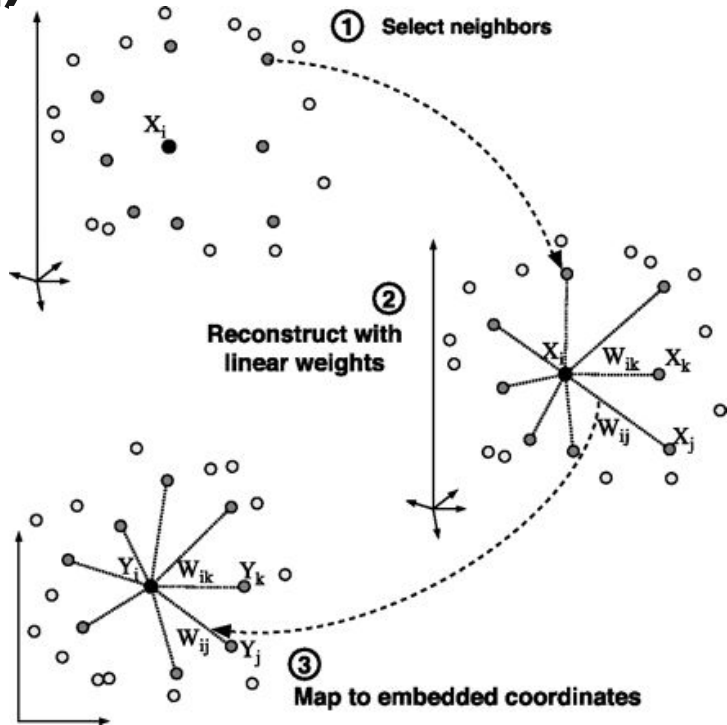
Kernel PCA

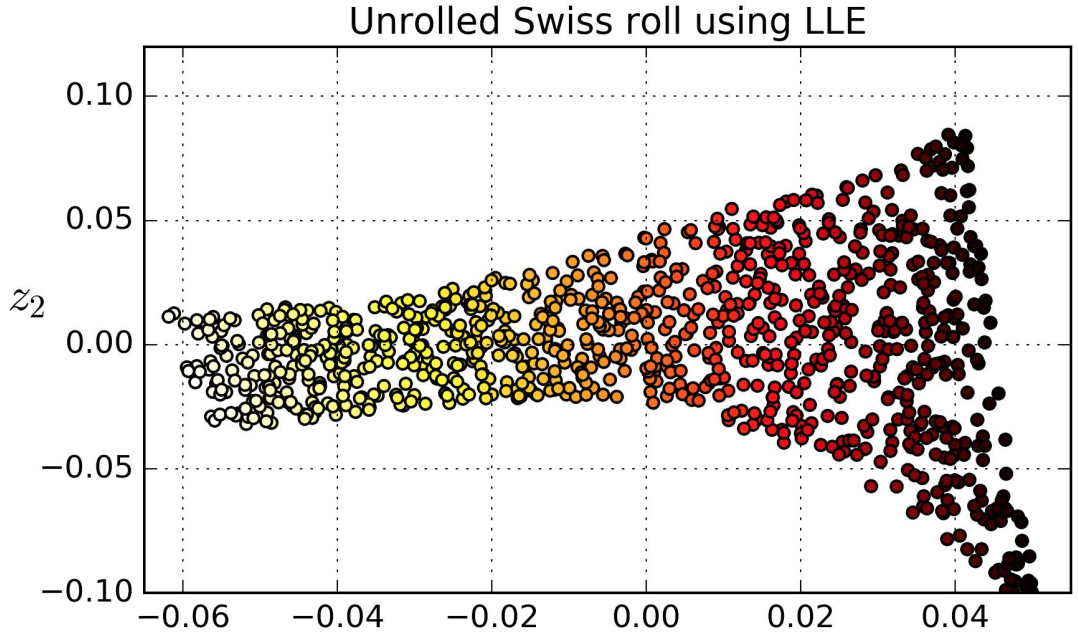
Apply kernel trick (a mathematical technique that implicitly maps instances into a very high-dimensional space) to PCA, making it possible to perform complex nonlinear projections for dimensionality reduction

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression())
    ])

param_grid = [{
        "kpca__gamma": np.linspace(0.03, 0.05, 10),
        "kpca__kernel": ["rbf", "sigmoid"]
    }]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

Locally Linear Embedding (LLE)

- A very powerful nonlinear dimensionality reduction technique
- Using Manifold Learning
- The distances between instances are locally well preserved, but not preserved on a larger scale






```
from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)
X_reduced = lle.fit_transform(X)
```