CHAPTER 8

Dimensionality Reduction

Chapter 8 explains **dimensionality reduction**: why high-dimensional data is hard to work with (curse of dimensionality), and how techniques like PCA, Kernel PCA, and manifold learning can reduce dimension for speed and visualization while trying to preserve structure.

Curse of dimensionality and motivation

In high dimensions, data becomes sparse and distances between points grow large, so models must extrapolate more and risk severe overfitting unless the dataset is enormous. Real-world data often lies near a much lower-dimensional **manifold**, because features are correlated or constrained (e.g., MNIST digits occupy a tiny subspace of all possible images). Dimensionality reduction can:

- Speed up training and inference, sometimes improve generalization by removing noise
- Enable **visualization** in 2D/3D (e.g., cluster plots)
  But it also adds pipeline complexity and usually discards some information, which can slightly hurt accuracy if overused.

Projection and manifold learning

One approach is **linear projection**: find a low-dimensional subspace (hyperplane) onto which data is projected, preserving as much variance as possible. For many datasets, points lie close to a flat subspace (e.g., 3D data lying near a plane) so projection works well. However, if the data lies on a curved manifold (e.g., the **Swiss roll**), simple projection squashes layers together; instead, **manifold learning** tries to "unroll" the manifold, preserving local structure rather than using a global flat plane. The **manifold assumption** says that learning on the intrinsic low-dimensional manifold is often easier, but the chapter cautions that decision boundaries are not always simpler after reduction; sometimes the original space is more convenient.

PCA (Principal Component Analysis)

PCA is the main linear dimensionality reduction tool. It:

- Centers data then finds **principal components**: orthogonal directions (axes) that maximize variance.

- The first component explains the most variance, the second the most remaining variance, etc., found via **SVD** decomposition $X = U\Sigma V^\top$, where columns of V are principal directions.
- To reduce to d dimensions, project onto the first d components: $X_{\text{proj}} = XW_d$, where W_d contains the top d eigenvectors.

Using Scikit-Learn's PCA, you can either specify n_components as an integer or as a variance ratio (e.g., n_components=0.95 to keep 95% of variance) and inspect explained_variance_ratio_ or its cumulative sum to choose d. PCA can also be used for **compression**: MNIST can be reduced from 784 to ~154 dimensions while preserving 95% variance, drastically shrinking storage and speeding SVM training, with only modest reconstruction error upon inverse_transform.

Variants include:

- **Randomized PCA** (svd_solver="randomized") which approximates top components much faster for large n, with complexity $O(md^2)$ when $d \ll n$.
- **Incremental PCA (IPCA)**, which processes mini-batches via partial_fit or memory-mapped arrays, suitable when the full dataset doesn't fit in RAM or for online scenarios.

Kernel PCA

Kernel PCA (kPCA) applies the kernel trick to PCA, allowing **nonlinear** dimensionality reduction. It uses kernels such as RBF, polynomial, or sigmoid to implicitly map data into a high-dimensional feature space and perform PCA there, often unrolling manifolds or preserving cluster structure better than linear PCA. With KernelPCA, you specify kernel and its hyperparameters (e.g., gamma for RBF) and can tune them **supervised** by wrapping kPCA in a pipeline before a classifier (e.g., Logistic Regression) and using GridSearchCV on downstream validation performance. Alternatively, one can minimize **reconstruction pre-image error** by approximating the inverse mapping and measuring how close reconstructed points are to originals, but this is more complex.

LLE and other nonlinear methods

Locally Linear Embedding (LLE) is a manifold learning method that preserves local linear relationships:

1. For each point, find its k nearest neighbors and compute weights that best reconstruct it as a linear combination of those neighbors.
2. Find low-dimensional coordinates that preserve these reconstruction weights as closely as possible.
   LLE is particularly good at unrolling manifolds like the Swiss roll, preserving local neighborhoods even if global distances distort somewhat. Its computational cost includes an $O(dm^2)$ term, so it scales poorly to very large datasets.

The chapter briefly surveys other techniques:

- **Random Projections**: use a random linear map that, by the Johnson–Lindenstrauss lemma, approximately preserves distances in lower dimensions.
- **MDS (Multidimensional Scaling)**: preserves pairwise distances.
- **Isomap**: preserves **geodesic** distances using a neighborhood graph.
- **t-SNE**: specialized for 2D/3D visualization, preserving local neighborhoods and separating clusters (popular for MNIST visualizations).
- **LDA (Linear Discriminant Analysis)**: supervised; finds directions that best separate class means and can be used as a supervised dimensionality reduction before a classifier.

For your Word summary, the figures to capture from Chapter 8 include: the hypercube illustrations that motivate the curse of dimensionality, the 3D-to-2D projection example, the Swiss roll and its “unrolled” versions via various methods, the PCA explained variance elbow plot, and the MNIST compression/reconstruction images