Chapter 8: Dimensionality Reduction

- Background
    - Millions of features: slow training
    - Remove unimportant info, merge correlated info
    - Still some info loss
    - Good for data viz
- The Curse of Dimensionality
    - Probabilities change in higher d: extreme outcomes more likely
    - Two pts in 1e6 cube = sqrt(1e6/6) average distance
    - High d data sets are sparse: lots of space in btwn
        - New instance likely to be far away from training
        - Overfitting goes up with higher d
    - Increasing density intractable solution: amount of data required to reach given level of density grows exponentially
- Main approaches for dimensionality reduction
    - Projection
        - Most training instances: non-uniform spread
            - Constant or highly-correlated features
            - Training instances will lie in lower d *subspace*
        - Projection not always best approach if subspace twists and turns; e.g. Swiss roll
    - Manifold Learning
        - 2d manifold: 2d shape that can be bent and twisted in higher dimensions
        - d-dimensional manifold part of n-dimensional space d<n
            - Locally resembles d-dimensional hyperplane
        - Manifold hypothesis: most real-world high-d datasets lie close to lower-d manifold
            - Degrees of freedom to generate data much lower than degrees in which you find that data
            - Assumption: task (classification/regression) simpler if expressed in lower-d space → NOT ALWAYS TRUE
- PCA
    - Background: PCA most popular dimensionality reduction algo
    - Preserving the variance
        - Need to choose right hyperplane before project training set onto lower-d
        - Choose one that preserves max variance: loses less info
        - Choose axis that minimizes mean squared distance btwn original and projection
    - Principal components
        - Identifies axis that accounts for largest amount of variance
            - Finds lines in remaining dimensions perpendicular to that line
        - For each principal component (PC), PCA finds zero-centered unit vector ([0,1]) pointing in direction of PC

    - Two opposing unit vectors line on same axis: hence direction returned by PCA is not stable, but will generally lie on same axes and plane
    - To find PC: use singular value decomposition: decomposes training matrix into matrix multiplication of three matrices: $U \Sigma V^T$
    - PCA assumes data centered around origin, sklearn takes care of that
  - Projecting down to d Dimensions
    - Once PCs identified → project dataset onto hyperplane defined by 1st d PCs
    - To project: multiply original matrix by first d columns of V: the W_d matrix
  - Using scikit-learn
    - Fits PCA transformed to dataset
    - components_ holds W_d transpose
  - Explained Variance Ratio
    - Proportion of dataset's variance that lies along each PC
  - Choosing the right number of dimensions
    - Choose number of dimensions that add up to large proportion of variance
    - Choose 2 if using PCA for data viz
    - pca = PCA(n_components=0.95)
  - PCA for Compression
    - After dimensionality reduction, training takes up less space
    - Reduction to 20% of original size is reasonable
    - To decompress apply inverse transform
      - Will lose some info since projection dropped 5% of variance
      - Reconstruction error: mean squared distance btwn original and reconstructed
  - Randomized PCA
    - Set svd_solver to "randomized"
    - Sklearn automatically uses if m or n > 500 & d < 80% of m or n
  - Incremental PCA
    - Split training into mini-batches
- Kernel PCA
  - Background:
    - Kernel trick: mapping to high dimensional space; linear decision boundary in high-dimensional space equivalent to complex non-linear boundary in original space
    - Kernel trick can be applied to Kernel PCA
  - Selecting a Kernel and Tuning hyperparameters
    - kPCA is an unsupervised learning algo → no obvious performance measure
    - Can use grid search to select kernel and hyperparameters with best performance
    - Other approach: select kernel & hyperparameters with lowest reconstruction error
      - Reconstruction not as easy: transformation mapped to infinite dimensional space which means cannot compute reconstruction point and thus reconstruction error

            - Solution: find point in original space close to reconstructed point (reconstruction pre-image) → measure preimage squared distance to original instance
            - Train supervised regression model with projected instances as training set and original instances as targets
                - Use fit_inverse_transform=True in sklearn
- LLE
    - Background: locally linear embedding → nonlinear dimensionality reduction
    - Manifold learning that does not rely on projections
    - Measures how each instance linearly relates to closest neighbor, then looks for low-d representation of training set where local relationships best preserved
    - Good at unrolling twisted manifolds
    - How it works:
        - For each instance identify closest neighbors
        - Reconstruct instance as linear function of neighbors
        - Map training instances onto d-dimensional space (where d < n) while preserving local relationships
        - Using local relationship to find min of squared distances btwn instance and neighbors in higher d
    - Algo doesn't scale well on large datasets
- Other dimensionality reduction techniques
    - Random projections
        - Projects stat to lower-d space using random linear projection
        - Quality of dimensionality reduction depends on number of instances
    - Multidimensional scaling
        - Reduces dimensionality trying to preserve distance btwn instances
    - Isomap
        - Connects each instance to nearest neighbors, reduces dimensionality while trying to preserve geodesic distance (number of nodes on shortest path btwn two nodes)
    - t-Distributed stochastic Neighbor embedding
        - Keep similar instances close, dissimilar instances apart
        - Mostly used for visualization
    - Linear discriminant analysis
        - Learns most discriminative axes btwn classed and then uses axes to define hyperplane on which to project data
        - Keeps classes as far apart as possible