CAI 4104/6108 – Machine Learning Engineering:

Unsupervised Learning

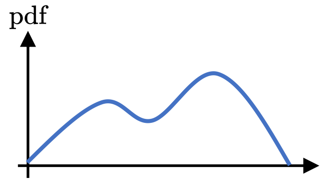
Prof. Vincent Bindschaedler

Spring 2024

Unsupervised Learning

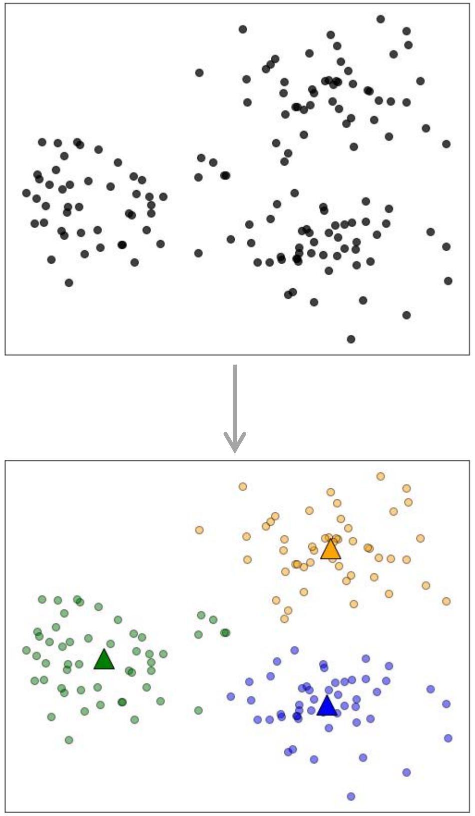
- Learning from unlabeled data (we must discover patterns in the data)

- Problems / Tasks
  - Density Estimation
    - Model the (unknown) probability distribution function (pdf) from which the data is drawn
  - Outlier Detection
    - Identify data points that do **not** belong in (i.e., are very different from) the (training) data
  - Clustering
    - Group similar data points together in "clusters"
  - Dimensionality Reduction (& Visualization)
    - Transform dataset into a low(er)-dimensional representation in a way that preserves useful information
  - Association Rule Mining
    - Discover interesting relationships between variables in the data



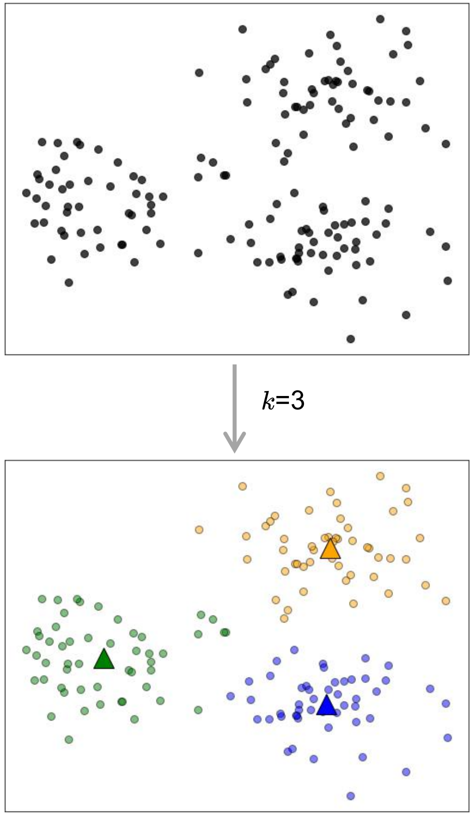
Clustering

- Goal: assign data points to clusters in the optimal way
  - What does optimal mean?
  - How many clusters?
- Dataset: Matrix $\mathbf{X}$ $(n \times m)$; no labels
- Clustering algorithms
  - k-means clustering
  - DBSCAN
  - Hierarchical clustering
- Q: Do we need a "holdout" set?
  - Yes! The clustering could overfit.
  - So we should train - test split or train - test - val split



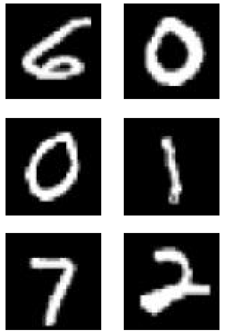
Clustering: K-Means

- K-Means Clustering
  - Greedy iterative algorithm to assign points to clusters
  - Input: dataset $\mathbf{X}$, integer k (hyperparameter — number of clusters)
  - Objective function: $L = \sum_i \; || \boldsymbol{x}_i - c(\boldsymbol{x}_i) ||^2$
    - Here $c(\boldsymbol{x}_j)$ is the centroid of the cluster that $\boldsymbol{x}_j$ is assigned to.
  - Algorithm:
    - Initialization: random centroid for each cluster
    - Do until no change in clustering:
      - Assignment: Assign each data point to the cluster with the closest centroid
      - Update: Recalculate centroids from points assigned to each cluster
- Q: How do we determine k?
  - Guess or try a bunch of different options
  - Heuristics: Silhouette method or Elbow method
  - Tune it as you would any other hyperparameter (if you have an objective metric to evaluate the quality of a clustering — e.g. co-membership test)
  - Or: Use a different algorithm that does not require specifying k (e.g., DBSCAN)



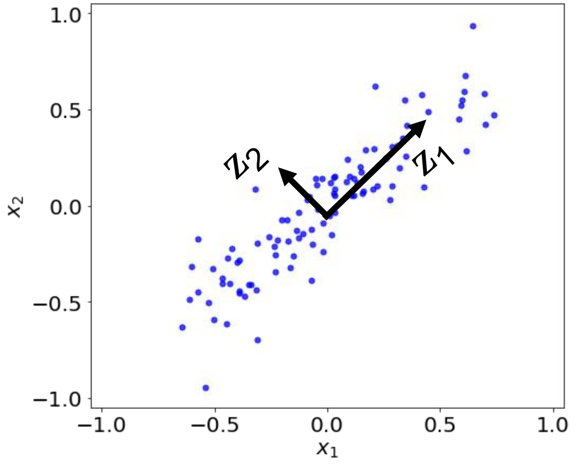
Dimensionality Reduction

- Curse of Dimensionality
  - Working with high dimensional data is challenging
  - For example: what do we do if each example in our dataset has millions of features?
    - Not only would training be slow, but model performance may suffer. Why?
  - What can we do?
    - Increase the amount of training data to compensate for the large number of features
      - Not feasible in practice (in most cases)
    - If the dimensionality of the data is artificial, then we should be able to reduce dimension without losing information
- Dimensionality Reduction Techniques
  - Projections (e.g., PCA)
    - Project the data into a lower dimensional subspace by exploiting feature correlations
  - Manifold Learning (e.g., LLE, Isomap)
    - Assume the data lies on some manifold. Model that manifold to reduce dimensionality
  - Note: another motivation for reducing dimensionality is visualization



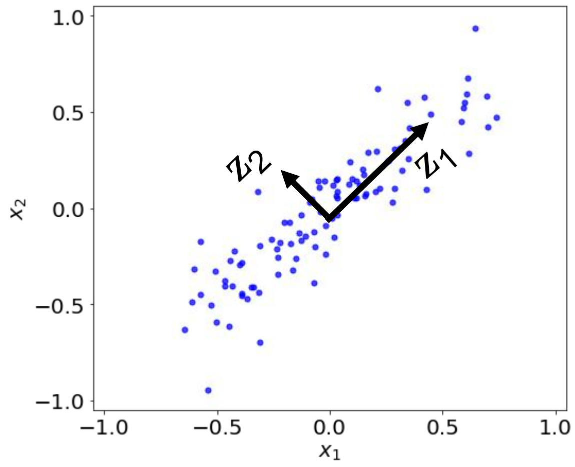
Principal Components Analysis (PCA)

- Principal Components Analysis (PCA)
  - Method for linear transformation onto a new system of coordinates
    - Invented by Pearson in 1901
    - The transformation is such that the principal components (coordinate vectors) capture the greatest amount of variance given the previous components
  - Algorithm:
    - Given data matrix $\mathbf{X}$ ($n \times m$)
      - Mean-center it: subtract the mean of each feature
    - Compute covariance matrix $\mathbf{X}^\top\mathbf{X}$ ($m \times m$)
    - The eigendecomposition of gives principal components
      - We can use Singular Value Decomposition (SVD)
      - Matrix $\mathbf{W}$ of eigenvectors is the transformation matrix (the i^{th} column is the i^{th} principal component)
      - Eigenvalue λ_i gives the variance of i^{th} principal component
  - Note: we can do PCA on the correlation matrix instead of covariance matrix



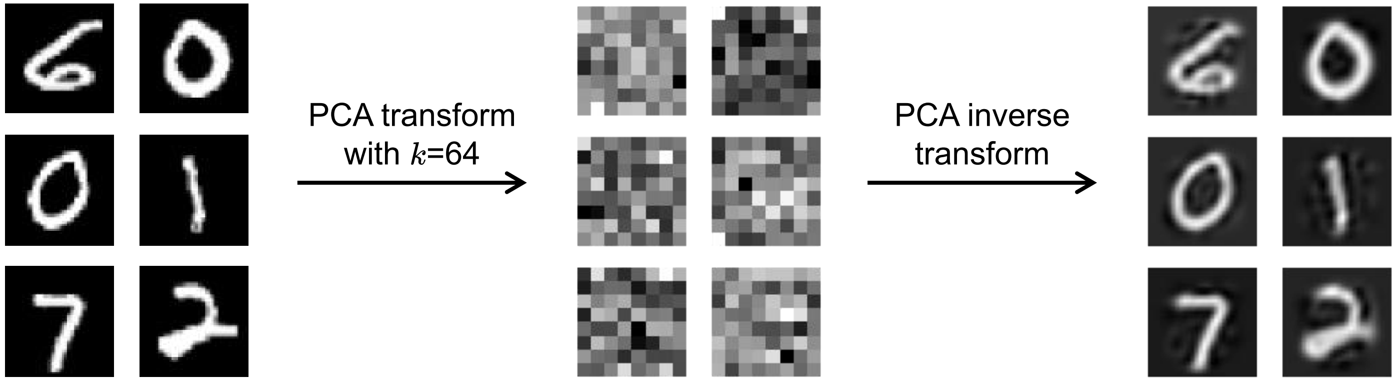
Principal Components Analysis (PCA)

- PCA Transformation / Dimensionality Reduction
  - Given data matrix $\mathbf{X}$ ($n \times m$), transformation matrix $\mathbf{W}$ ($m \times m$), and eigenvalues λ
    - $\mathbf{Z} = \mathbf{X}\ \mathbf{W}$ is the transformed data onto the PCA coordinate system
  - If we want to reduce dimensionality, we can keep only the first k principal components
    - Think of k as a hyperparameter
    - Let $\mathbf{W}_k$ is the transformation matrix with only the first k columns ($m \times k$)
    - Transformed data: $\mathbf{Z}_k = \mathbf{X}\ \mathbf{W}_k$ [here is a $n \times k$ matrix]
  - How much of the variance is explained?
    - Variance explained: $(\lambda_1+\lambda_2+...+\lambda_k) / \sum_i \lambda_i$
  - How do we determine k?
    - Same way we tune other hyperparameters
    - Or: set k to explain a fixed portion of the variance (e.g., 95%)



Principal Components Analysis (PCA)

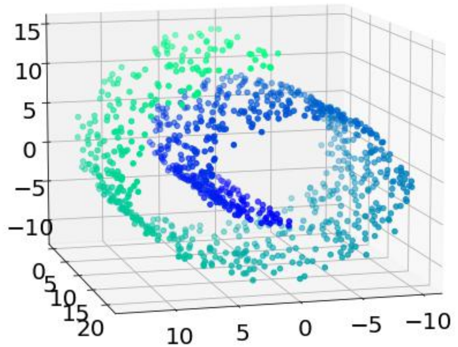
- PCA Transformation / Dimensionality Reduction
  - Inverse transformation: can we transform the data back to the original space?
    - Yes: $\mathbf{X}' = \mathbf{Z}_k \mathbf{W}_k^{\mathsf{T}}$ where $\mathbf{W}_k^{\mathsf{T}}$ is a $k \times m$ matrix
    - Here: $\mathbf{X}'$ is a $(n \times m)$ matrix (note: some information was lost in the transformation unless $k = m$)
  - What does this actually look like?

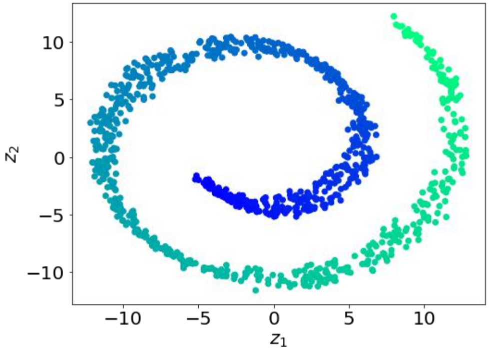


Kernel PCA

- PCA is a linear decomposition/transformation
  - What if we need *non-linear* dimensionality reduction?
    - We can use the kernel trick (the same one we used for SVM); this is called Kernel PCA

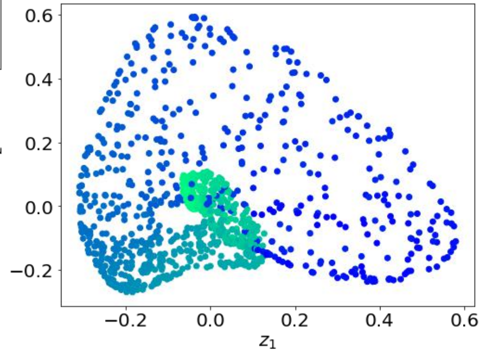
Swiss Roll

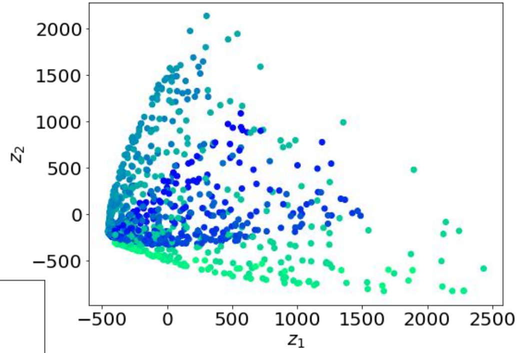




Linear kernel (= PCA)

RBF kernel (γ=0.05)



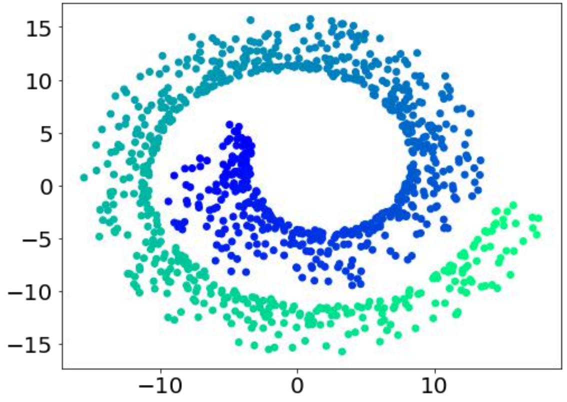


Polynomial kernel (degree 3)

Manifold Learning

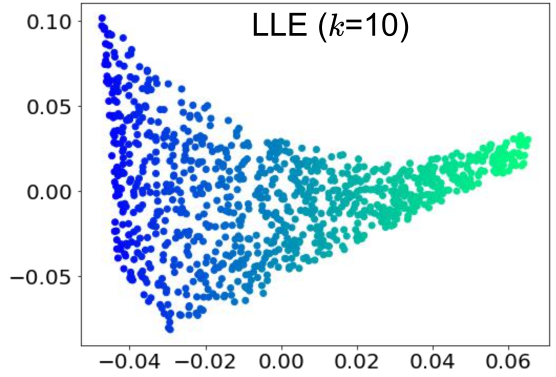
- Manifold Learning
  - Given a dataset $\boldsymbol{X}$={$\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_n$} embedded in $\mathbb{R}^m$ (each point $\boldsymbol{x}_i \in \mathbb{R}^m$)
  - Find a mapping of the dataset $\boldsymbol{X}$ into a dataset $\boldsymbol{Z}$={$\boldsymbol{z}_1$, $\boldsymbol{z}_2$, ..., $\boldsymbol{z}_n$} embedded in $\mathbb{R}^p$ ($\boldsymbol{z}_i \in \mathbb{R}^p$) for some integer $p < m$
  - Such that (informally) $\boldsymbol{Z}$ preserves the local geometry of $\boldsymbol{X}$
    - For example if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are close (according to some distance metric), then $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ are also close
    - So really we want to preserve the neighborhood structure

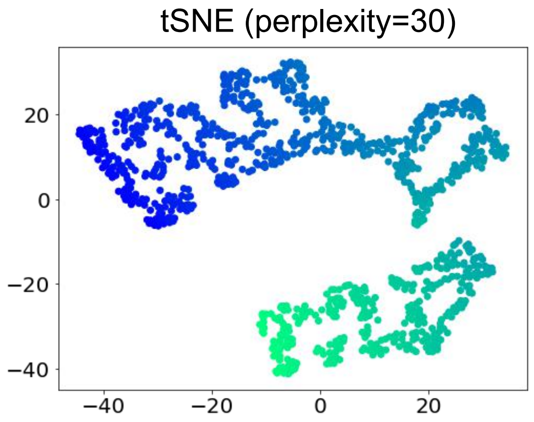
- Multidimensional Scaling (MDS)
  - Compute the Euclidean distance d_{ij} between any two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$
  - We want to minimize: $\sum_{i<j} [\, || \boldsymbol{z}_i - \boldsymbol{z}_j || - d_{ij}]^2$
    - The minimization here is over $\boldsymbol{z}_1$, $\boldsymbol{z}_2$, ..., $\boldsymbol{z}_n$
  - What if we don't care about (approximately) preserving large distances?
    - Then we could ask to only preserve distances only if two points are neighbors



Manifold Learning

- Algorithms
  - Different algorithms aim to preserve different "local" properties
  - Multidimensional Scaling (MDS)
    - Preserve distances between points
  - Locally Linear Embedding (LLE)
    - Express each data point as a linear combination of its closest neighbors
  - Isomap
    - Form a graph where data points are connected to their closest neighbors
    - Aims to preserve geodesic distance (i.e., shortest path distance)
  - t-distributed Stochastic Neighbor Embedding (t-SNE)
    - Tries to keep similar data points close together, dissimilar data points far apart
    - Mostly used for visualization
- Q: Does feature scaling matter?
  - Yes, we should rescale the data (to ensure all features have the scale)
  - Why? Methods are based on nearest-neighbors search





Next Time

- Friday (2/23): Exercise

- Upcoming:
  - Homework 2 due 2/23