

CS 1810 Spring 2026 Section 7

Unsupervised Learning

1 Introduction

In supervised learning we are given labeled data pairs $(\mathbf{x}_i, y_i)$, and the goal is to learn a mapping from inputs to outputs. However, this is not always our goal. For example, when you're scrounging for shows to watch on Netflix, you probably notice that it has a list of recommended items. In generating this list, Netflix uses something called a *recommender system*, where given a large dataset of previous reviews/watch history, it predicts which other shows you would rate highly. In this case, Netflix is working under the *unsupervised learning* setting, where it's given a collection of data points (your previous watch history) but without true labels (your true enjoyment of new shows).

Because there are no labels, a main goal of many unsupervised methods is typically to discover underlying representations of the data. Thus, we typically need to define a notion of similarity, structure, or information content (e.g., geometry, density, invariances), and evaluation is often contextual, i.e. we assess fit, stability, or usefulness rather than prediction error.

2 Clustering

2.1 Problem Setup

Given data points $\{\mathbf{x}_i\}_{i=1}^N$, the goal of clustering is to partition the points into K groups so that points in the same group are similar or related. The output is a set of assignments $c_i \in \{1, \dots, K\}$. Using the same Netflix recommendation example, a clustering algorithm might be used to identify other viewers to whom you are similar. The idea is that you all would then share similar tastes, so if these similar viewers tend to enjoy a certain show, that show would also get recommended to you. It is worth noting that clustering is often an intermediate step in a larger pipeline, not the final goal itself.

2.2 K -Means

2.2.1 Objective

The most widely used partition-based clustering method is K -means. The idea is to represent each cluster by a single prototype, called a centroid, and to assign each data point to the cluster whose centroid is nearest. Formally, we seek assignments $\{c_i\}_{i=1}^N$ and centroids $\{\boldsymbol{\mu}_j\}_{j=1}^K$ that minimize the within-cluster sum of squares:

$$\{c_i^*\}_{i=1}^N, \{\boldsymbol{\mu}_j^*\}_{j=1}^K = \arg \min \sum_{i=1}^N \|\mathbf{x}_i - \boldsymbol{\mu}_{c_i}\|_2^2. \quad (1)$$

This objective can be interpreted as a compression: it measures how well each data point is approximated by its cluster's centroid. The problem is non-convex (in fact NP-hard in general), so in practice we settle for local optima.

2.2.2 Lloyd's Algorithm

The standard algorithm for solving the K -means problem is Lloyd's algorithm, which alternates between two steps:

1. **Assignment step.** Fix the centroids and assign each point to the nearest centroid:

$$c_i \leftarrow \arg \min_{j \in \{1, \dots, K\}} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|_2^2, \quad i = 1, \dots, N.$$

Because the overall sum decomposes over data points, each assignment can be optimized independently. For fixed centroids this step is globally optimal.

2. **Update step.** Fix the assignments and recompute each centroid as the mean of its assigned points:

$$\boldsymbol{\mu}_k \leftarrow \frac{1}{n_k} \sum_{i: c_i = k} \mathbf{x}_i, \quad k = 1, \dots, K,$$

where $n_k = |\{i : c_i = k\}|$. To see why the mean is optimal, expand $\sum_{i: c_i = k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2^2$, differentiate with respect to $\boldsymbol{\mu}_k$, and set the gradient to zero:

$$-2 \sum_{i: c_i = k} \mathbf{x}_i + 2n_k \boldsymbol{\mu}_k = 0 \implies \boldsymbol{\mu}_k = \frac{1}{n_k} \sum_{i: c_i = k} \mathbf{x}_i.$$

Both steps can only decrease (or leave unchanged) the objective (1), so the algorithm converges monotonically to a fixed point. Since there are finitely many possible assignments, convergence is guaranteed in a finite number of iterations, though the solution may be only a local minimum.

2.2.3 Geometric Interpretation

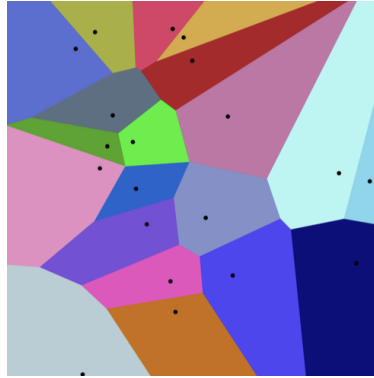


Figure 1: Credit: Wikimedia. Sample visualization of K -means cluster boundary shapes.

For a fixed set of centroids $\{\boldsymbol{\mu}_j\}$, the assignment rule $c_i = \arg \min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|_2^2$ partitions the input space into a Voronoi tessellation: each region consists of all points closer to a given centroid than to any other. The boundary between the regions for centroids $\boldsymbol{\mu}_a$ and $\boldsymbol{\mu}_b$ is the perpendicular bisector of the segment $\boldsymbol{\mu}_a \boldsymbol{\mu}_b$, so cluster boundaries are always linear. This means K -means implicitly assumes convex, roughly spherical clusters (see Figure 1).

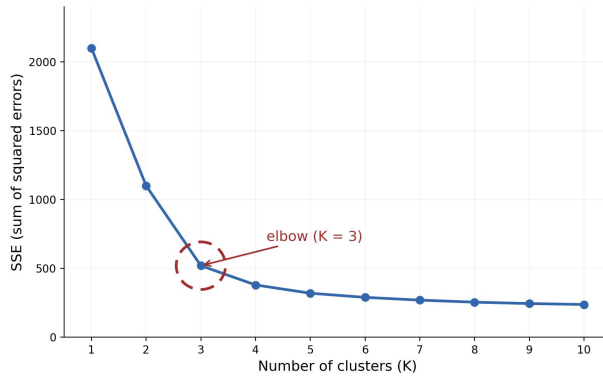


Figure 2: Plot the objective function vs K . Find the value of K where increasing K further only yields marginal improvements, i.e. an “elbow”.

2.2.4 Practical Considerations

1. **Choosing K :** More clusters always reduce the objective, but too many clusters defeat the purpose of summarization. Common heuristics include looking for an “elbow” in a plot of the objective versus K (Figure 2), assessing cluster stability across runs, or using downstream task constraints.
2. **Computational cost:** Each iteration of Lloyd’s algorithm costs $O(NKd)$ for the assignment step and $O(Nd)$ for the update step. Both steps parallelize naturally over data points and clusters, respectively.
3. **Initialization:** We saw above that different initializations can lead to the algorithm reaching different local minima. The K -means++ algorithm (see Section 2.4) addresses this by choosing initial centroids that are spread apart, providing much better guarantees than random initialization.
4. **Feature scaling:** If one coordinate has a much larger range than others, it will dominate the objective function. Standardizing the features mitigates this problem.
5. **Outliers:** Because the objective uses squared distances, outliers exert a disproportionate pull on centroids. Robust variants such as K -medoids (which uses actual data points as the “centroids” and minimizes the sum of distances rather than squared distances) can alleviate this issue.

2.2.5 Failure Modes

K -means works best when the true clusters are approximately spherical with similar variance. It can fail when clusters have non-spherical shapes (e.g., crescent-shaped clusters), when clusters have very different variances (the algorithm tends to split large clusters and merge small ones), or when K is misspecified.

2.2.6 Exercises

Exercise 1. Consider the 1D data $\{1, 2, 8, 9, 10\}$ with $K = 2$ and initial centroids $\mu_1 = 2, \mu_2 = 9$. Perform one full iteration of Lloyd’s algorithm (assign, then update). What are the new centroids?

Has the algorithm converged?

Solution: *Data:* $\{1, 2, 8, 9, 10\}$, $K = 2$, initial centroids $\mu_1 = 2, \mu_2 = 9$.

Assignment step. We assign each point to the nearest centroid:

x_i	$(x_i - \mu_1)^2$	$(x_i - \mu_2)^2$	assign
1	1	64	μ_1
2	0	49	μ_1
8	36	1	μ_2
9	49	0	μ_2
10	64	1	μ_2

So cluster 1 is $\{1, 2\}$ and cluster 2 is $\{8, 9, 10\}$.

Update step. Recompute each centroid as the mean of its assigned points:

$$\mu'_1 = \frac{1+2}{2} = 1.5, \quad \mu'_2 = \frac{8+9+10}{3} = 9.$$

Has the algorithm converged? We need to check whether a second assignment step would change anything. With $\mu_1 = 1.5$ and $\mu_2 = 9$, the midpoint between the two centroids is $(1.5 + 9)/2 = 5.25$. All points in $\{1, 2\}$ are below 5.25 and all points in $\{8, 9, 10\}$ are above it, so the assignments would not change. The algorithm has converged after one iteration.

Exercise 2. The K -means update step sets each centroid to the *mean* of its assigned points. Suppose we replaced the squared Euclidean objective $\sum_i \|\mathbf{x}_i - \boldsymbol{\mu}_{c_i}\|_2^2$ with the sum of (unsquared) Euclidean distances $\sum_i \|\mathbf{x}_i - \boldsymbol{\mu}_{c_i}\|_2$. Would the optimal centroid for a cluster still be the mean of its points? Why or why not?

Solution: No, the optimal centroid would no longer be the mean. To see why, consider the two objectives for a single cluster C :

Squared objective: We minimize $f(\boldsymbol{\mu}) = \sum_{i \in C} \|\mathbf{x}_i - \boldsymbol{\mu}\|_2^2$. Taking the gradient and setting it to zero:

$$\nabla_{\boldsymbol{\mu}} f = -2 \sum_{i \in C} (\mathbf{x}_i - \boldsymbol{\mu}) = 0 \implies \boldsymbol{\mu}^* = \frac{1}{|C|} \sum_{i \in C} \mathbf{x}_i,$$

which is the mean.

Unsquared objective: We instead minimize $g(\boldsymbol{\mu}) = \sum_{i \in C} \|\mathbf{x}_i - \boldsymbol{\mu}\|_2$. Taking the gradient (for $\mathbf{x}_i \neq \boldsymbol{\mu}$):

$$\nabla_{\boldsymbol{\mu}} g = - \sum_{i \in C} \frac{\mathbf{x}_i - \boldsymbol{\mu}}{\|\mathbf{x}_i - \boldsymbol{\mu}\|_2}.$$

Setting this to zero gives

$$\sum_{i \in C} \frac{\mathbf{x}_i - \boldsymbol{\mu}}{\|\mathbf{x}_i - \boldsymbol{\mu}\|_2} = 0 \implies \boldsymbol{\mu} = \frac{\sum_{i \in C} \mathbf{x}_i / \|\mathbf{x}_i - \boldsymbol{\mu}\|_2}{\sum_{i \in C} 1 / \|\mathbf{x}_i - \boldsymbol{\mu}\|_2}.$$

This is a *weighted* mean where each point is weighted by the inverse of its distance to $\boldsymbol{\mu}$ —and $\boldsymbol{\mu}$ itself appears on both sides, so there is no closed-form solution. The minimizer of the unsquared objective is the *geometric median*, not the arithmetic mean. Intuitively, because the unsquared distance does not penalize large deviations as heavily as the squared distance, the geometric median is more robust to outliers than the mean.

Exercise 3. Prove that it is not possible for the within-cluster sum of squares to increase between consecutive iterations of Lloyd's algorithm.

Solution: Let $J(\{c_i\}, \{\mu_j\}) = \sum_{i=1}^N \|\mathbf{x}_i - \mu_{c_i}\|_2^2$ denote the K -means objective. Suppose at the start of some iteration we have assignments $\{c_i\}$ and centroids $\{\mu_j\}$. We will show that each step of Lloyd's algorithm can only decrease (or leave unchanged) the objective.

Assignment step. Holding the centroids $\{\mu_j\}$ fixed, we update each assignment to

$$c'_i = \arg \min_{j \in \{1, \dots, K\}} \|\mathbf{x}_i - \mu_j\|_2^2.$$

Since the sum decomposes over data points, each term $\|\mathbf{x}_i - \mu_{c'_i}\|_2^2 \leq \|\mathbf{x}_i - \mu_{c_i}\|_2^2$ by construction (we chose the minimizing cluster). Therefore

$$J(\{c'_i\}, \{\mu_j\}) = \sum_{i=1}^N \|\mathbf{x}_i - \mu_{c'_i}\|_2^2 \leq \sum_{i=1}^N \|\mathbf{x}_i - \mu_{c_i}\|_2^2 = J(\{c_i\}, \{\mu_j\}).$$

Update step. Holding the new assignments $\{c'_i\}$ fixed, we update each centroid to

$$\mu'_k = \frac{1}{n_k} \sum_{i: c'_i=k} \mathbf{x}_i.$$

For each cluster k , the function $\mu \mapsto \sum_{i: c'_i=k} \|\mathbf{x}_i - \mu\|_2^2$ is a strictly convex quadratic, and the mean is its unique global minimizer (as shown by setting the gradient to zero). Therefore $\sum_{i: c'_i=k} \|\mathbf{x}_i - \mu'_k\|_2^2 \leq \sum_{i: c'_i=k} \|\mathbf{x}_i - \mu_k\|_2^2$ for each k , and summing over all clusters:

$$J(\{c'_i\}, \{\mu'_j\}) \leq J(\{c'_i\}, \{\mu_j\}).$$

Thus we have

$$J(\{c'_i\}, \{\mu'_j\}) \leq J(\{c'_i\}, \{\mu_j\}) \leq J(\{c_i\}, \{\mu_j\}).$$

The objective is non-increasing across each full iteration. Since there are finitely many possible assignment vectors $\{c_i\}$ (at most K^N) and the objective strictly decreases whenever an assignment changes, the algorithm must terminate in a finite number of steps.

2.3 Hierarchical Agglomerative Clustering (HAC)

2.3.1 Motivation

K -means requires specifying the number of clusters K in advance and produces only a single partition. Hierarchical agglomerative clustering (HAC) takes a different approach: it builds an entire hierarchy of clusterings, represented as a tree called a dendrogram. One can then obtain a solution for any desired number of clusters by "cutting" the dendrogram at the appropriate height. This is useful when there is no prior notion of a reasonable K .

2.3.2 Algorithm

HAC is a bottom-up (agglomerative) procedure. It requires specifying a linkage function $D(A, B)$ that measures the distance between two clusters A and B .

1. **Initialize:** At the bottom layer of the dendrogram, each data point forms its own singleton cluster, yielding N clusters.
2. Repeat until one cluster remains:
 - (a) Find the pair of clusters (A, B) with the smallest linkage distance $D(A, B)$.
 - (b) Merge A and B into a single cluster $A \cup B$; record the merge distance as the height in the dendrogram.
 - (c) Update the distances from the new cluster $A \cup B$ to all remaining clusters according to the linkage function.

2.3.3 Linkage Functions

The linkage function is the main inductive bias in HAC; different choices lead to qualitatively different clusterings:

- **Single linkage:** $D(A, B) = \min_{a \in A, b \in B} d(a, b)$. This uses the distance between the closest pair of points in the two clusters. It can discover elongated, non-convex cluster shapes, but is sensitive to noise and “bridge” points that can chain together clusters that should be separate (called the chaining effect).
- **Complete linkage:** $D(A, B) = \max_{a \in A, b \in B} d(a, b)$. This uses the distance between the farthest pair of points in the two clusters. It controls cluster diameter, i.e. produces compact clusters, avoiding the chaining effect.
- **Average linkage:** $D(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$. This averages all pairwise distances and represents a compromise between single and complete linkage, often yielding stable results.
- **Ward linkage:** $D(A, B) = \text{SSE}(A \cup B) - \text{SSE}(A) - \text{SSE}(B)$, where $\text{SSE}(C) = \sum_{x \in C} \|x - \bar{x}_C\|_2^2$. This merges the pair that causes the smallest increase in total within-cluster variance, producing compact clusters similar to those from K -means.

2.3.4 K -Means vs. HAC

K -means is simple, fast, and scalable, and produces easily interpretable centroids; however, it requires K to be specified in advance, is sensitive to initialization and outliers, and assumes roughly spherical clusters. HAC produces a full hierarchy without needing K in advance and can accommodate different shape biases through the linkage function, but it is more computationally expensive (naive implementations are $O(N^3)$), cannot undo early merges, and is sensitive to noise (especially with single linkage).

2.3.5 Evaluating Clusterings

Because clustering is unsupervised, there is no single ground-truth criterion. Common internal metrics assess compactness (small within-cluster distances), separation (large between-cluster distances), and stability (consistency across runs or subsamples). These criteria are often in tension with one another. When labels become available, external consistency metrics (e.g., whether clusters correlate with true classes) can also be used.

2.3.6 Exercises

Exercise 1. Consider 1D points $\{0, 2, 4, 5, 8\}$ using Euclidean distance. After the first merge, suppose the current clusters are $\{0, 2\}$, $\{4, 5\}$, $\{8\}$. Compute the distance between $\{0, 2\}$ and $\{4, 5\}$ under (a) single linkage and (b) complete linkage. Which linkage merges these two clusters next? Which merges $\{4, 5\}$ with $\{8\}$ next?

Solution:

- (a) **Single Linkage.** Single linkage uses the minimum distance between any pair of points across clusters:

$$d_{\text{single}}(C_1, C_2) = \min_{x \in C_1, y \in C_2} |x - y| = \min\{|0 - 4|, |0 - 5|, |2 - 4|, |2 - 5|\} = \min\{4, 5, 2, 3\} = 2$$

- (b) **Complete Linkage.** Complete linkage uses the maximum distance between any pair of points across clusters:

$$d_{\text{complete}}(C_1, C_2) = \max_{x \in C_1, y \in C_2} |x - y| = \max\{|0 - 4|, |0 - 5|, |2 - 4|, |2 - 5|\} = \max\{4, 5, 2, 3\} = 5$$

- (c) **Which linkage merges C_1 and C_2 next?**

- **Single linkage:** $d_{\text{single}}(C_2, C_3) = \min\{|4 - 8|, |5 - 8|\} = \min\{4, 3\} = 3$.
Since $d_{\text{single}}(C_1, C_2) = 2 < 3 = d_{\text{single}}(C_2, C_3)$, single linkage merges $\{0, 2\}$ with $\{4, 5\}$ first.
- **Complete linkage:** $d_{\text{complete}}(C_2, C_3) = \max\{|4 - 8|, |5 - 8|\} = \max\{4, 3\} = 4$.
Since $d_{\text{complete}}(C_2, C_3) = 4 < 5 = d_{\text{complete}}(C_1, C_2)$, complete linkage also merges $\{4, 5\}$ with $\{8\}$ first.

Exercise 2. Explain in one or two sentences why single linkage is susceptible to “chaining” while complete linkage is not. Give a simple geometric configuration where single linkage would merge two clearly separated groups.

Solution: Single linkage only requires one pair of close points to merge two clusters, so a chain of intermediate points (each close to the next) can link together two well-separated groups even if most points in those groups are far apart. Complete linkage avoids this because it requires all pairs of points to be close before merging, preventing a few “bridge” points from pulling distinct clusters together.

Consider two groups:

$$A = \{0, 1\}, \quad B = \{10, 11\}$$

with a chain of intermediate points $\{3, 5, 7\}$ between them. Single linkage would successively merge along the chain (since consecutive points are only distance 2 apart), eventually joining A and B into one cluster despite the groups being clearly separated. Complete linkage would resist this because $d_{\text{complete}}(A, B) = 11$, which remains large regardless of intermediate points.

2.4 Bonus: K -Means++

Random initialization of centroids can place them close together, leading to slow convergence and poor local minima. The K -means++ algorithm addresses this by choosing initial centroids that are spread out:

1. Choose the first centroid μ_1 uniformly at random from the data $\{\mathbf{x}_n\}_{n=1}^N$.
2. For $k = 2, \dots, K$: for each data point $\mathbf{x}_n$, compute $d_n = \min_{k' < k} \|\mathbf{x}_n - \mu_{k'}\|_2$, the distance to the nearest already-chosen centroid. Then sample μ_k from the data with probability proportional to d_n^2 .

Points that are far from all existing centroids are more likely to be chosen, encouraging diversity among the initial centers. After initialization, Lloyd's algorithm proceeds as usual. This simple modification provides a provable guarantee: the expected objective is within an $O(\log K)$ factor of the global optimum, which is much better than random initialization. In practice, K -means++ is the default in most libraries and often reduces the number of restarts needed.

3 Dimension Reduction: PCA

In this next section, we will introduce Principal Component Analysis (PCA) by first deriving its optimal solution based on the reconstruction loss and then connecting it to the variance-maximization perspective of PCA. This section is very math-heavy, so we would like to emphasize that it is more important to understand the intuition behind PCA than every step in its derivation. In particular, you should understand the equivalence between minimizing reconstruction loss and maximizing variance of the projected data (Section 3.4.3), as well as the connection to eigenvalue decomposition (Section 3.5). Section 3.7 provides a nice summary of how PCA is typically executed.

3.1 Motivation

Clustering assigns data to discrete groups, but sometimes the structure in the data is better described by a few continuous latent factors. For example, images of faces vary smoothly with pose and lighting; gene expression profiles are driven by a small number of biological programs; and genomic data often reflects geographic ancestry through a handful of principal axes.

When the data are high-dimensional but the features are highly correlated, some features are mostly noise, or the effective variation is driven by only a few latent factors, it is natural to seek a low-dimensional representation which captures the important structure. Principal component analysis (PCA) is the canonical method for this task, and it is useful for compression, denoising, visualization, and simplifying downstream modeling.

3.2 Setup

Given data $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)} \in \mathbb{R}^D$, PCA seeks a set of $K \ll D$ vectors $\mathbf{v}_1, \dots, \mathbf{v}_K \in \mathbb{R}^D$ such that each data point can be well approximated as a linear combination

$$\mathbf{x} \approx z_1 \mathbf{v}_1 + \dots + z_K \mathbf{v}_K.$$

In the language of linear algebra, the vectors $\mathbf{v}_k$ are basis vectors (the principal components), and the scalars z_k are the coordinates of $\mathbf{x}$ in this new basis. If the approximation is good, we can replace the D original coordinates with only K latent coordinates, i.e. the latent coordinates should

be representative of the data.

The key question is: what makes a particular basis good? The standard basis $\mathbf{e}_1 = [1, 0, \dots]^\top$, $\mathbf{e}_2 = [0, 1, \dots]^\top$, etc., need not align with the directions along which the data actually vary. PCA finds a rotated basis that does align with the data's shape.

3.3 The PCA Optimization Problem

Let $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_K] \in \mathbb{R}^{D \times K}$ denote the basis matrix and let $\mathbf{z}^{(n)} \in \mathbb{R}^K$ denote the code for the n -th data point. PCA minimizes the average reconstruction error:

$$J(\mathbf{V}, \mathbf{Z}) = \frac{1}{N} \sum_{n=1}^N \left\| \mathbf{x}^{(n)} - \mathbf{V} \mathbf{z}^{(n)} \right\|_2^2. \quad (2)$$

Note the structural similarity to K -means: both minimize a sum of squared distances, but in K -means the reconstruction μ_{c_n} is one of K discrete prototypes, whereas in PCA the reconstruction $\mathbf{V} \mathbf{z}^{(n)}$ is a continuous linear combination.

3.3.1 Non-uniqueness and the Orthonormality Constraint

The minimizer of (2) is not unique without further constraints: if $(\mathbf{V}, \mathbf{Z})$ is a solution, then for any invertible $K \times K$ matrix $\mathbf{Q}$, the pair $(\tilde{\mathbf{V}}, \tilde{\mathbf{Z}}) = (\mathbf{V} \mathbf{Q}, \mathbf{Q}^{-1} \mathbf{Z})$ achieves the same objective value, since $\tilde{\mathbf{V}} \tilde{\mathbf{z}}^{(n)} = \mathbf{V} \mathbf{Q} \mathbf{Q}^{-1} \mathbf{z}^{(n)} = \mathbf{V} \mathbf{z}^{(n)}$.

To resolve this ambiguity, we impose the constraint that $\mathbf{V}$ is orthonormal: $\mathbf{V}^\top \mathbf{V} = \mathbf{I}_K$. This removes arbitrary scaling and skewing, yields orthogonal directions (like the typical coordinate axes), and makes the components easier to interpret.

3.3.2 Mean-Centering

If the data are not centered around the origin, PCA can produce misleading results: the first component may simply point toward the data's mean rather than capturing any interesting variation. Therefore, we always mean-center the data before applying PCA:

$$\tilde{\mathbf{x}}^{(n)} = \mathbf{x}^{(n)} - \bar{\mathbf{x}}, \quad \bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}^{(n)}.$$

With centered data, PCA is able to capture variation around the mean. From this point on, we assume the data are centered and write $\tilde{\mathbf{x}}$ for the centered points.

3.4 Deriving the Solution

We now derive the PCA solution in three steps: (1) find the optimal code for a fixed basis, (2) simplify the objective, and (3) interpret the result.

3.4.1 Step 1: Optimal Code for Fixed Basis

For a fixed orthonormal basis $\mathbf{V}$, we seek the code $\mathbf{z}^{(n)}$ that minimizes

$$\left\| \tilde{\mathbf{x}}^{(n)} - \mathbf{V}\mathbf{z}^{(n)} \right\|_2^2 = \left\| \tilde{\mathbf{x}}^{(n)} \right\|_2^2 - 2\mathbf{z}^{(n)\top} \mathbf{V}^\top \tilde{\mathbf{x}}^{(n)} + \left\| \mathbf{z}^{(n)} \right\|_2^2,$$

where the third term uses $\mathbf{V}^\top \mathbf{V} = \mathbf{I}_K$. Differentiating with respect to $\mathbf{z}^{(n)}$ and setting the gradient to zero gives

$$-2\mathbf{V}^\top \tilde{\mathbf{x}}^{(n)} + 2\mathbf{z}^{(n)} = 0 \implies \mathbf{z}^{(n)} = \mathbf{V}^\top \tilde{\mathbf{x}}^{(n)}.$$

That is, the optimal code is the orthogonal projection of the centered data point onto the column space of $\mathbf{V}$. The orthonormality of $\mathbf{V}$ is what makes this formula so clean.

3.4.2 Step 2: Simplifying the Objective

Substituting $\mathbf{z}^{(n)} = \mathbf{V}^\top \tilde{\mathbf{x}}^{(n)}$ back into the objective yields an objective that depends only on $\mathbf{V}$:

$$J(\mathbf{V}) = \frac{1}{N} \sum_{n=1}^N \left\| \tilde{\mathbf{x}}^{(n)} - \mathbf{V}\mathbf{V}^\top \tilde{\mathbf{x}}^{(n)} \right\|_2^2.$$

Since $\mathbf{V}\mathbf{V}^\top$ is an orthogonal projection matrix, we can expand:

$$J(\mathbf{V}) = \frac{1}{N} \sum_{n=1}^N \tilde{\mathbf{x}}^{(n)\top} (\mathbf{I} - \mathbf{V}\mathbf{V}^\top) \tilde{\mathbf{x}}^{(n)} = \frac{1}{N} \sum_{n=1}^N \left\| \tilde{\mathbf{x}}^{(n)} \right\|_2^2 - \frac{1}{N} \sum_{n=1}^N \tilde{\mathbf{x}}^{(n)\top} \mathbf{V}\mathbf{V}^\top \tilde{\mathbf{x}}^{(n)}.$$

The first term is a constant (it does not depend on $\mathbf{V}$). Expanding the second term using $\mathbf{V}\mathbf{V}^\top = \sum_{k=1}^K \mathbf{v}_k \mathbf{v}_k^\top$ gives

$$\frac{1}{N} \sum_{n=1}^N \tilde{\mathbf{x}}^{(n)\top} \mathbf{V}\mathbf{V}^\top \tilde{\mathbf{x}}^{(n)} = \sum_{k=1}^K \mathbf{v}_k^\top \left(\frac{1}{N} \sum_{n=1}^N \tilde{\mathbf{x}}^{(n)} \tilde{\mathbf{x}}^{(n)\top} \right) \mathbf{v}_k = \sum_{k=1}^K \mathbf{v}_k^\top \boldsymbol{\Sigma} \mathbf{v}_k,$$

where $\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^N \tilde{\mathbf{x}}^{(n)} \tilde{\mathbf{x}}^{(n)\top}$ is the empirical covariance matrix.

Therefore, minimizing reconstruction error is equivalent to maximizing the sum of projected variances:

$$\min_{\mathbf{V}^\top \mathbf{V} = \mathbf{I}_K} \frac{1}{N} \sum_{n=1}^N \left\| \tilde{\mathbf{x}}^{(n)} - \mathbf{V}\mathbf{z}^{(n)} \right\|_2^2 \iff \max_{\mathbf{V}^\top \mathbf{V} = \mathbf{I}_K} \sum_{k=1}^K \mathbf{v}_k^\top \boldsymbol{\Sigma} \mathbf{v}_k. \quad (3)$$

3.4.3 Step 3: Statistical Interpretation

The equivalence (3) gives us a nice intuition. For any unit direction $\mathbf{v}$, the scalar projection of a data point is $z = \mathbf{v}^\top \tilde{\mathbf{x}}$, and the variance of the projected data is

$$\text{Var}(z) = \frac{1}{N} \sum_{n=1}^N (\mathbf{v}^\top \tilde{\mathbf{x}}^{(n)})^2 = \mathbf{v}^\top \boldsymbol{\Sigma} \mathbf{v}.$$

Thus PCA finds orthonormal directions that **maximize the variance of the projected data**, i.e. PC1 is the direction of maximum variance; PC2 is the direction orthogonal to PC1 with the largest remaining variance; and so on.

Equivalently, PCA **minimizes reconstruction error**: it finds the K -dimensional subspace that best approximates the data in the least-squares sense. These two views yield exactly the same solution.

3.5 Solving PCA: Eigendecomposition

To find the first principal component ($K = 1$), we solve the constrained optimization problem

$$\max_{\mathbf{v}^\top \mathbf{v} = 1} \mathbf{v}^\top \Sigma \mathbf{v}.$$

Forming the Lagrangian $\mathcal{L}(\mathbf{v}, \lambda) = \mathbf{v}^\top \Sigma \mathbf{v} - \lambda(\mathbf{v}^\top \mathbf{v} - 1)$ and setting the gradient to zero gives

$$2\Sigma\mathbf{v} - 2\lambda\mathbf{v} = 0 \implies \Sigma\mathbf{v} = \lambda\mathbf{v}.$$

This is an eigenvalue equation: $\mathbf{v}$ must be an eigenvector of Σ , and λ is the corresponding eigenvalue. Since we want to maximize $\mathbf{v}^\top \Sigma \mathbf{v} = \lambda \mathbf{v}^\top \mathbf{v} = \lambda$, we should pick the eigenvector with the largest eigenvalue. Refer to Section 0 for a review of eigendecomposition.

It can be shown that the same argument extends to the remaining components: the k -th principal component is the eigenvector of Σ with the k -th largest eigenvalue, subject to orthogonality with all previously chosen components.

3.5.1 Eigenvalue Interpretation

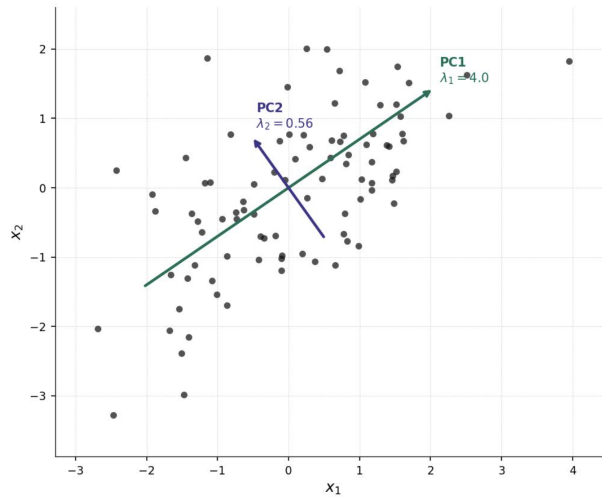


Figure 3: Datapoints with the first two principle components (eigenvectors) and their corresponding eigenvalues labeled.

Since $\Sigma \mathbf{v}_k = \lambda_k \mathbf{v}_k$ and $\|\mathbf{v}_k\| = 1$, we have

$$\text{Var}(z_k) = \mathbf{v}_k^\top \Sigma \mathbf{v}_k = \lambda_k \mathbf{v}_k^\top \mathbf{v}_k = \lambda_k.$$

So each eigenvalue λ_k is exactly the variance captured (or “explained”) along its corresponding principal direction. Larger eigenvalues correspond to more important components (Figure 3).

3.6 PCA via SVD

Eigendecomposition and singular value decomposition are tightly related. If $\tilde{\mathbf{X}} \in \mathbb{R}^{N \times D}$ is the centered data matrix (rows are data points) with SVD $\tilde{\mathbf{X}} = \mathbf{U} \mathbf{S} \mathbf{V}^\top$, then $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} = \mathbf{V} \mathbf{S}^2 \mathbf{V}^\top$. Since $\Sigma = \frac{1}{N} \tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$, the right singular vectors $\mathbf{V}$ are eigenvectors of Σ , and the eigenvalues are $\lambda_k = s_k^2/N$. In practice, PCA is usually computed via the SVD of $\tilde{\mathbf{X}}$ rather than by forming and decomposing Σ explicitly, since the SVD is numerically more stable.

3.7 The PCA Recipe

Putting everything together, the practical procedure for PCA is:

1. Form the data matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$.
2. Center the data: $\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^\top$, where $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}^{(n)}$.
3. Compute the empirical covariance matrix $\Sigma = \frac{1}{N} \tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$.
4. Compute PCA by either eigendecomposition of Σ (giving $\Sigma = \mathbf{V}\Lambda\mathbf{V}^\top$) or SVD of $\tilde{\mathbf{X}}$ (giving $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$).
5. Keep the top K directions $\mathbf{V}_K = [\mathbf{v}_1, \dots, \mathbf{v}_K]$ and project: $\mathbf{Z} = \tilde{\mathbf{X}}\mathbf{V}_K$.

3.8 Interpreting PCA Output

PCA produces three complementary pieces of output:

- **Scores/codes.** The coordinates of each data point in the PC basis, $\mathbf{z}^{(n)} = \mathbf{V}^\top \tilde{\mathbf{x}}^{(n)} \in \mathbb{R}^K$. These describe the data points in the low-dimensional space.
- **Loadings.** The entries of each principal component $\mathbf{v}_k$. The j -th entry $(\mathbf{v}_k)_j$ tells us how much original feature j contributes to the k -th PC, providing interpretability.
- **Explained variance.** The eigenvalue λ_k gives the variance captured by the k -th component. The explained variance ratio $\text{EVR}_k = \lambda_k / \sum_{j=1}^D \lambda_j$ gives the fraction of total variance captured, and the cumulative explained variance $\text{CumEVR}(K) = \sum_{k=1}^K \lambda_k / \sum_{j=1}^D \lambda_j$ measures how much variance the first K components retain.

3.8.1 Choosing K

Common strategies for choosing the number of components include keeping enough PCs to explain 90–95% of the total variance, or looking for an “elbow” in the scree plot (a plot of λ_k versus k). There is no universal threshold; the right choice depends on the application.

3.9 The Compression Interpretation

The PCA objective can also be viewed as an encoder-decoder scheme. The encoder maps $\mathbf{x} \in \mathbb{R}^D$ to $\mathbf{z} = \mathbf{V}^\top \tilde{\mathbf{x}} \in \mathbb{R}^K$, and the decoder maps $\mathbf{z}$ back to $\hat{\mathbf{x}} = \mathbf{V}\mathbf{z} = \mathbf{V}\mathbf{V}^\top \tilde{\mathbf{x}} \in \mathbb{R}^D$. We want a small code ($K \ll D$) and an accurate reconstruction ($\|\mathbf{x} - \hat{\mathbf{x}}\|^2$ small). PCA is the best linear encoder-decoder under squared error. See Section 6 Notes for more details on this.

3.10 Failure Modes of PCA

PCA is a powerful and general tool, but it does have limitations:

1. **Outliers.** Because PCA is based on variance (a squared quantity), outliers can disproportionately affect the principal directions.
2. **Nonlinearity.** PCA finds a linear subspace. If the data lie on a curved manifold, a linear subspace will miss the structure.

3. **Task relevance.** PCA finds the directions of maximum variance, which need not be the directions most useful for a downstream task. The direction of greatest variance may be orthogonal to the direction that best separates classes.

3.11 PCA vs. Clustering

PCA and clustering represent two complementary types of structure. Clustering seeks discrete groups: it outputs labels or centroids, and each point is explained by a cluster identity. It works best when data form separated clusters. PCA seeks continuous directions of variation: it outputs coordinates in a subspace, and each point is explained by a few continuous factors. It works best when data lie near a low-dimensional linear structure. The same dataset may benefit from both methods.

3.12 Exercises

Exercise 1. Suppose all data points are translated by a constant vector $\mathbf{a}$, i.e., $\mathbf{x}^{(n)} \mapsto \mathbf{x}^{(n)} + \mathbf{a}$. Does this change the principal components? Does it change the scores? Explain briefly.

Solution:

Principal components: No change. PCA operates on the centered data $\tilde{\mathbf{x}}^{(n)} = \mathbf{x}^{(n)} - \bar{\mathbf{x}}$. If every point is shifted by $\mathbf{a}$, the new mean becomes $\bar{\mathbf{x}}' = \bar{\mathbf{x}} + \mathbf{a}$, so the centered data is

$$\tilde{\mathbf{x}}'^{(n)} = (\mathbf{x}^{(n)} + \mathbf{a}) - (\bar{\mathbf{x}} + \mathbf{a}) = \mathbf{x}^{(n)} - \bar{\mathbf{x}} = \tilde{\mathbf{x}}^{(n)}.$$

The centered data is unchanged, so the covariance matrix and its eigenvectors (the principal components) remain the same.

Scores: No change. The scores are $\mathbf{z}^{(n)} = \mathbf{V}^\top \tilde{\mathbf{x}}^{(n)}$. Since the centered data $\tilde{\mathbf{x}}^{(n)}$ is unchanged, the scores are also unchanged.

Exercise 2. You run PCA on 100-dimensional data and keep $K = 3$ components. The eigenvalues are $\lambda_1 = 50$, $\lambda_2 = 30$, $\lambda_3 = 10$, and the remaining 97 eigenvalues sum to 10. What fraction of the total variance is captured? If you reconstruct a data point $\hat{\mathbf{x}} = \mathbf{V}_3 \mathbf{V}_3^\top \tilde{\mathbf{x}}$, is $\hat{\mathbf{x}}$ guaranteed to equal $\tilde{\mathbf{x}}$? Why or why not?

Solution: The total variance is the sum of all eigenvalues:

$$\text{Total variance} = 50 + 30 + 10 + 10 = 100.$$

The variance captured by the top 3 components is:

$$\text{Captured variance} = \lambda_1 + \lambda_2 + \lambda_3 = 50 + 30 + 10 = 90.$$

Therefore, the fraction captured is:

$$\frac{90}{100} = \boxed{90\%}.$$

The reconstruction $\hat{\mathbf{x}} = \mathbf{V}_3 \mathbf{V}_3^\top \tilde{\mathbf{x}}$ is generally not equal to $\tilde{\mathbf{x}}$.

The matrix $\mathbf{V}_3 \mathbf{V}_3^\top$ is a projection onto the 3-dimensional subspace spanned by the top 3 principal components. Since $\tilde{\mathbf{x}}$ lives in a 100-dimensional space and generally has nonzero components

along the other 97 eigenvector directions, projecting onto a 3D subspace discards information. Specifically:

$$\hat{\mathbf{x}} = \sum_{k=1}^3 (\mathbf{v}_k^\top \tilde{\mathbf{x}}) \mathbf{v}_k,$$

which omits the components $(\mathbf{v}_k^\top \tilde{\mathbf{x}}) \mathbf{v}_k$ for $k = 4, \dots, 100$. Equality $\hat{\mathbf{x}} = \tilde{\mathbf{x}}$ holds only if $\tilde{\mathbf{x}}$ already lies entirely within the span of $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$, which is not guaranteed.

Exercise 3. In the derivation, we showed that the optimal code is $\mathbf{z}^{(n)} = \mathbf{V}^\top \tilde{\mathbf{x}}^{(n)}$. This relied on $\mathbf{V}^\top \mathbf{V} = \mathbf{I}_K$. Suppose we dropped the orthonormality constraint. Would the formula $\mathbf{z} = \mathbf{V}^\top \tilde{\mathbf{x}}$ still give the optimal code? What would change?

Solution: No, the formula $\mathbf{z} = \mathbf{V}^\top \tilde{\mathbf{x}}$ would no longer give the optimal code if $\mathbf{V}$ is not orthonormal.

In PCA, we seek the code $\mathbf{z}$ that minimizes reconstruction error $\|\tilde{\mathbf{x}} - \mathbf{V}\mathbf{z}\|^2$. Taking the gradient with respect to $\mathbf{z}$ and setting it to zero:

$$\nabla_{\mathbf{z}} \|\tilde{\mathbf{x}} - \mathbf{V}\mathbf{z}\|^2 = -2\mathbf{V}^\top (\tilde{\mathbf{x}} - \mathbf{V}\mathbf{z}) = 0 \implies \mathbf{V}^\top \mathbf{V}\mathbf{z} = \mathbf{V}^\top \tilde{\mathbf{x}}.$$

The optimal code is therefore:

$$\mathbf{z}^* = (\mathbf{V}^\top \mathbf{V})^{-1} \mathbf{V}^\top \tilde{\mathbf{x}}.$$

When $\mathbf{V}$ is orthonormal, $\mathbf{V}^\top \mathbf{V} = \mathbf{I}_K$, so this simplifies to $\mathbf{z}^* = \mathbf{V}^\top \tilde{\mathbf{x}}$. But without orthonormality, we must use the pseudoinverse $(\mathbf{V}^\top \mathbf{V})^{-1} \mathbf{V}^\top$ instead of just $\mathbf{V}^\top$.

What changes:

- The encoding requires computing $(\mathbf{V}^\top \mathbf{V})^{-1}$, which is more expensive.
- The columns of $\mathbf{V}$ may be correlated or have different norms, so $\mathbf{V}^\top \tilde{\mathbf{x}}$ no longer correctly weights the projections.
- The elegant interpretation of scores as independent projections onto orthogonal directions is lost.