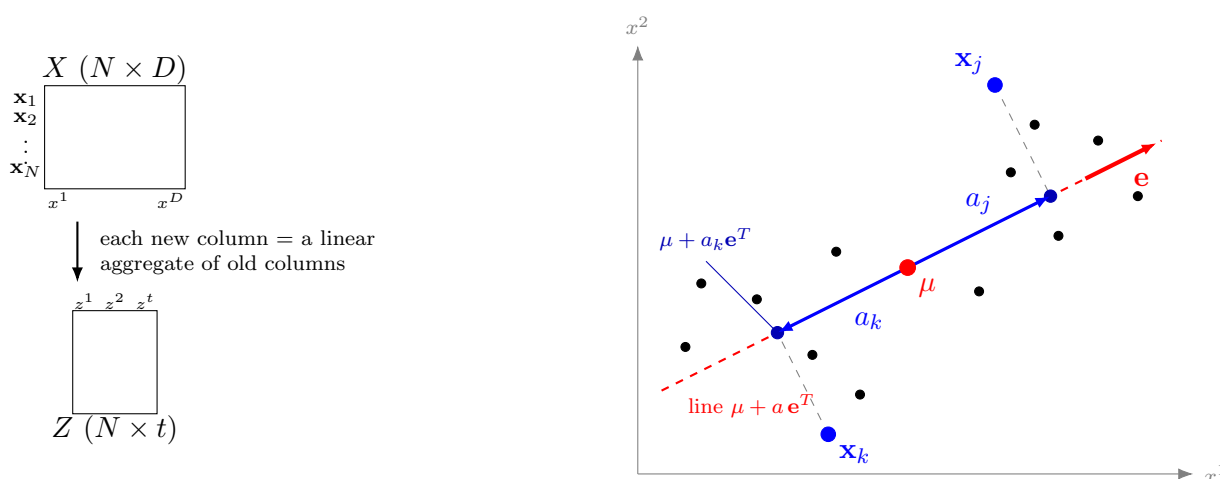


Principal Component Analysis (PCA) and Feature Representations

Lecture Notes Consolidated with Claude AI — CS6140 Machine Learning

1 The Goal: A Better Set of Columns

Our data is the usual $N \times D$ matrix X : one row $\mathbf{x}_k = (x_k^1, \dots, x_k^D)$ per datapoint, one column per feature. We want a *new* data matrix Z , of size $N \times t$, whose columns are new features built from the old ones:



Left: PCA builds new columns from the old ones. Right ($D = 2, t = 1$): each datapoint $\mathbf{x}_k$ is replaced by its projection $\mu + a_k \mathbf{e}^T$ on a line through the mean μ with unit direction $\mathbf{e}$; the single new feature is the signed coordinate $a_k = (\mathbf{x}_k - \mu) \mathbf{e}$ along that line. The best line follows the longest stretch of the data, so the projections lose as little as possible (gray dashes).

Each new feature is a *linear aggregate* of the old ones: $z^1 = (\mathbf{x} - \mu) \mathbf{e}_1 = \sum_j e_1^j (x^j - \mu^j)$ for some weight (direction) vector $\mathbf{e}_1$. What makes PCA's columns special is *how* they are chosen:

- The **first** new column z^1 is the *best* single linear aggregate: the one that keeps the most information about the data (made precise in §3).
- The **second** column z^2 is the next-best aggregate *that carries nothing already in z^1* (it is uncorrelated with z^1 ; its direction $\mathbf{e}_2$ is perpendicular to $\mathbf{e}_1$). The third is next-best given the first two, and so on (§4).
- **How many** such columns exist? Exactly as many as there are independent (perpendicular) directions in which the data actually varies — the rank of the covariance matrix, never more than D (§5). Past that point, new columns would carry no information at all.
- In practice we often keep a *smaller* t than that theoretical number: the last columns carry so little variation that they are mostly noise, a downstream classifier may work better (and faster) with fewer inputs, or we simply want $t = 2, 3$ to plot the data (§5.4).

Key point: The goal of PCA is not “make $t < D$ ” as such. It is “re-express the data in a sequence of columns, each one the best remaining linear aggregate, each one independent of the previous ones.” Dimensionality reduction is what you get by *stopping early* in that sequence. As we will derive, the directions $\mathbf{e}_1, \mathbf{e}_2, \dots$ are exactly the eigenvectors of the data’s covariance matrix, sorted by eigenvalue.

2 Setup: Centering and the Covariance Matrix

- **Mean** (a $1 \times D$ row vector): $\mu = \frac{1}{N} \sum_{k=1}^N \mathbf{x}_k$. **Centered data:** $\tilde{\mathbf{x}}_k = \mathbf{x}_k - \mu$, stacked into $\tilde{X}$ ($N \times D$, every column has mean 0).
- **Covariance matrix** ($D \times D$):

$$\Sigma = \frac{1}{N} \sum_{k=1}^N (\mathbf{x}_k - \mu)^T (\mathbf{x}_k - \mu) = \frac{1}{N} \tilde{X}^T \tilde{X}, \quad \Sigma_{ij} = \frac{1}{N} \sum_k (x_k^i - \mu^i)(x_k^j - \mu^j). \quad (1)$$

Each term $(\mathbf{x}_k - \mu)^T (\mathbf{x}_k - \mu)$ is a $(D \times 1)(1 \times D)$ matrix, so Σ is a sum of N matrices. On the diagonal, Σ_{jj} is the variance of feature j ; off the diagonal, $\Sigma_{ij} = \Sigma_{ji}$ measures how features i and j vary together (linear correlation, unnormalized).

- **Direction:** a unit vector $\mathbf{e}$ ($D \times 1$, $\|\mathbf{e}\|^2 = \mathbf{e}^T \mathbf{e} = 1$). The **projection** of datapoint k onto the line through μ with direction $\mathbf{e}$ is the scalar $a_k = (\mathbf{x}_k - \mu)\mathbf{e}$ — its new coordinate on that line.

Remark. Some texts use $\frac{1}{N-1}$, or no factor at all (the “scatter matrix”), instead of $\frac{1}{N}$. A constant factor scales every eigenvalue but does not change any eigenvector, so the PCA directions are identical under all three conventions.

3 The First Direction

Start with $t = 1$: represent every datapoint by a single number, its position along one line through μ . Which line? Visually, the one along the “longest stretch” of the data. Two natural ways to make this precise lead to the same answer.

3.1 View 1: Smallest Reconstruction Error

Each datapoint is replaced by a point on the line, $\mu + a_k \mathbf{e}^T$ ($1 \times D$). The total squared reconstruction error (SSE) is

$$J(a_1, \dots, a_N, \mathbf{e}) = \sum_{k=1}^N \left\| \mu + a_k \mathbf{e}^T - \mathbf{x}_k \right\|^2 = \sum_k \left\| a_k \mathbf{e}^T - (\mathbf{x}_k - \mu) \right\|^2 \quad (2)$$

$$= \sum_k a_k^2 \underbrace{\|\mathbf{e}\|^2}_{=1} - 2 \sum_k a_k (\mathbf{x}_k - \mu) \mathbf{e} + \sum_k \|\mathbf{x}_k - \mu\|^2. \quad (3)$$

Best coordinates for a fixed line. Setting $\partial J / \partial a_k = 2a_k - 2(\mathbf{x}_k - \mu)\mathbf{e} = 0$ gives

$$a_k = (\mathbf{x}_k - \mu)\mathbf{e}, \quad (4)$$

i.e. the best coordinate is exactly the perpendicular projection. *Best line.* Substituting a_k back, the middle term becomes $-2 \sum_k a_k^2$, so

$$J(\mathbf{e}) = - \sum_k ((\mathbf{x}_k - \mu)\mathbf{e})^2 + \sum_k \|\mathbf{x}_k - \mu\|^2 = - \sum_k \mathbf{e}^T (\mathbf{x}_k - \mu)^T (\mathbf{x}_k - \mu) \mathbf{e} + \text{const} \quad (5)$$

$$= -N \mathbf{e}^T \Sigma \mathbf{e} + \text{const}, \quad (6)$$

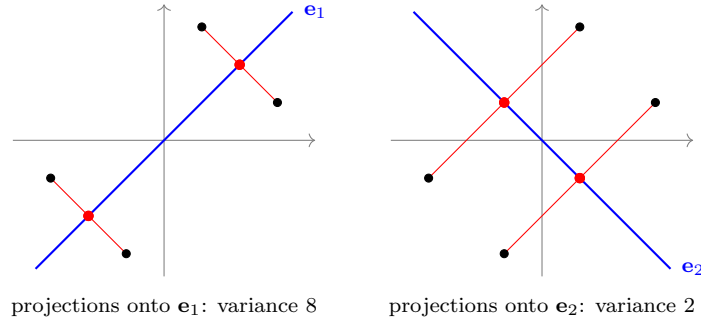
using $((\mathbf{x}_k - \mu)\mathbf{e})^2 = \mathbf{e}^T (\mathbf{x}_k - \mu)^T (\mathbf{x}_k - \mu) \mathbf{e}$ (a scalar times itself, written as “transpose times itself”) and pulling $\mathbf{e}$ out of the sum. The last term does not depend on $\mathbf{e}$, so

$$\min_{\mathbf{e}} J(\mathbf{e}) \iff \max_{\mathbf{e}} \mathbf{e}^T \Sigma \mathbf{e}. \quad (7)$$

3.2 View 2: Largest Variance of the Projections

The new coordinates $a_1, \dots, a_N$ are the values of the new feature. Their mean is $\frac{1}{N} \sum_k (\mathbf{x}_k - \mu)\mathbf{e} = (\mu - \mu)\mathbf{e} = 0$, so their variance is

$$\text{Var}[a] = \frac{1}{N} \sum_k a_k^2 = \frac{1}{N} \sum_k \mathbf{e}^T (\mathbf{x}_k - \mu)^T (\mathbf{x}_k - \mu) \mathbf{e} = \mathbf{e}^T \Sigma \mathbf{e}. \quad (8)$$



The four centered points of the worked example (§7). Projected onto $\mathbf{e}_1$ (left) the red points stay spread out, $a_k = \pm 2.83$; onto $\mathbf{e}_2$ (right) they bunch together, $a_k = \pm 1.41$ (each red dot holds two points).

Key point: “Lose the least when projecting” (View 1) and “keep the most spread” (View 2) are the *same objective*: total spread = spread kept + spread lost, and the total is fixed. The best single linear aggregate is the one with maximum variance.

3.3 Solving It: Lagrange Multiplier

Without a constraint, $\mathbf{e}^T \Sigma \mathbf{e}$ grows forever by scaling $\mathbf{e}$ up, so we fix the length — only the *direction* matters:

$$\max_{\mathbf{e}} \mathbf{e}^T \Sigma \mathbf{e} \quad \text{subject to} \quad \mathbf{e}^T \mathbf{e} = 1. \quad (9)$$

This is a **constrained optimization** problem. The standard tool is the **Lagrangian**: fold the constraint into the objective with a new variable λ (the Lagrange multiplier),

$$L(\mathbf{e}, \lambda) = \mathbf{e}^T \Sigma \mathbf{e} - \lambda (\mathbf{e}^T \mathbf{e} - 1), \quad (10)$$

and look for points where its gradient is zero. (Geometric intuition: at the constrained optimum, the gradient of the objective is parallel to the gradient of the constraint; λ is the ratio between them.) Using $\partial(\mathbf{e}^T \mathbf{A} \mathbf{e}) / \partial \mathbf{e} = 2\mathbf{A} \mathbf{e}$ for symmetric $\mathbf{A}$:

$$\frac{\partial L}{\partial \mathbf{e}} = 2\Sigma \mathbf{e} - 2\lambda \mathbf{e} = 0 \implies \boxed{\Sigma \mathbf{e} = \lambda \mathbf{e}} \quad (11)$$

Multiplying an arbitrary vector by Σ usually changes its direction; Equation 11 says that for the optimal $\mathbf{e}$ it does *not* — Σ only stretches it, by the factor λ . That is the definition of an **eigenvector** $\mathbf{e}$ of Σ with **eigenvalue** λ . Which eigenvector? Plug Equation 11 back into the objective:

$$\text{Var}[a] = \mathbf{e}^T \Sigma \mathbf{e} = \mathbf{e}^T (\lambda \mathbf{e}) = \lambda \mathbf{e}^T \mathbf{e} = \lambda. \quad (12)$$

Key point: The variance captured along an eigenvector *is* its eigenvalue. So the best first direction $\mathbf{e}_1$ is the eigenvector of Σ with the **largest** eigenvalue λ_1 , and the first new feature $z^1 = (\mathbf{x} - \mu) \mathbf{e}_1$ has variance λ_1 .

4 The Next Directions: Best Remaining, Independent of the Previous

For the second column we want the next-best aggregate, but one that carries nothing already captured by the first: its direction must be perpendicular to $\mathbf{e}_1$. Add that as a second constraint, with its own multiplier γ :

$$\max_{\mathbf{e}} \mathbf{e}^T \Sigma \mathbf{e} \quad \text{s.t.} \quad \mathbf{e}^T \mathbf{e} = 1, \quad \mathbf{e}_1^T \mathbf{e} = 0; \quad L = \mathbf{e}^T \Sigma \mathbf{e} - \lambda(\mathbf{e}^T \mathbf{e} - 1) - \gamma \mathbf{e}_1^T \mathbf{e}. \quad (13)$$

$$\frac{\partial L}{\partial \mathbf{e}} = 2\Sigma \mathbf{e} - 2\lambda \mathbf{e} - \gamma \mathbf{e}_1 = 0. \quad (14)$$

Multiply Equation 14 on the left by $\mathbf{e}_1^T$:

$$2\mathbf{e}_1^T \Sigma \mathbf{e} - 2\lambda \underbrace{\mathbf{e}_1^T \mathbf{e}}_{=0} - \gamma \underbrace{\mathbf{e}_1^T \mathbf{e}_1}_{=1} = 0, \quad \text{and} \quad \mathbf{e}_1^T \Sigma \mathbf{e} = (\Sigma \mathbf{e}_1)^T \mathbf{e} = \lambda_1 \mathbf{e}_1^T \mathbf{e} = 0 \quad (15)$$

(Σ is symmetric, so $\mathbf{e}_1^T \Sigma = (\Sigma \mathbf{e}_1)^T$). Hence $\gamma = 0$, Equation 14 reduces to $\Sigma \mathbf{e} = \lambda \mathbf{e}$ again, and the best choice is the eigenvector $\mathbf{e}_2$ with the **second-largest** eigenvalue λ_2 . The same argument, with one extra multiplier per previous direction, gives the third, fourth, $\dots$: the j -th PCA direction is the eigenvector with the j -th largest eigenvalue.

The new columns are uncorrelated. For two new features $z^i = (\mathbf{x} - \mu) \mathbf{e}_i$ and $z^j = (\mathbf{x} - \mu) \mathbf{e}_j$ ($i \neq j$), both with mean 0:

$$\text{Cov}[z^i, z^j] = \frac{1}{N} \sum_k ((\mathbf{x}_k - \mu) \mathbf{e}_i) ((\mathbf{x}_k - \mu) \mathbf{e}_j) = \mathbf{e}_i^T \Sigma \mathbf{e}_j = \lambda_j \mathbf{e}_i^T \mathbf{e}_j = 0. \quad (16)$$

Key point: In the new coordinates the covariance matrix is *diagonal*: $\text{diag}(\lambda_1, \lambda_2, \dots)$. Each new column carries its own share of the variation and none of the others' — this is the precise sense of “independent of the previous ones.” (Uncorrelated, i.e. *linearly* independent information. Full statistical independence is a stronger property — see ICA, §10.3.)

5 How Many Directions? Spectral Decomposition of Σ

5.1 Σ Is Special

- **Symmetric:** $\Sigma_{ij} = \Sigma_{ji}$.
- **Positive semidefinite:** for any $\mathbf{v}$, $\mathbf{v}^T \Sigma \mathbf{v} = \frac{1}{N} \mathbf{v}^T \tilde{X}^T \tilde{X} \mathbf{v} = \frac{1}{N} \|\tilde{X} \mathbf{v}\|^2 \geq 0$. In particular every eigenvalue is ≥ 0 (take $\mathbf{v}$ an eigenvector: $\lambda = \mathbf{v}^T \Sigma \mathbf{v} \geq 0$) — a variance cannot be negative.

Symmetric matrices have a **spectral decomposition**. Let $E = [\mathbf{e}_1 \mathbf{e}_2 \cdots \mathbf{e}_D]$ ($D \times D$, eigenvectors as columns) and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_D)$ with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$. Then

$$\Sigma = E \Lambda E^T = \lambda_1 \mathbf{e}_1 \mathbf{e}_1^T + \lambda_2 \mathbf{e}_2 \mathbf{e}_2^T + \cdots + \lambda_D \mathbf{e}_D \mathbf{e}_D^T. \quad (17)$$

Why. (1) Eigenvectors with different eigenvalues are orthogonal: $\lambda_i \mathbf{e}_i^T \mathbf{e}_j = (\Sigma \mathbf{e}_i)^T \mathbf{e}_j = \mathbf{e}_i^T \Sigma \mathbf{e}_j = \lambda_j \mathbf{e}_i^T \mathbf{e}_j$, so $(\lambda_i - \lambda_j) \mathbf{e}_i^T \mathbf{e}_j = 0$. (Eigenvectors sharing an eigenvalue can be chosen orthogonal.) Normalized, $E^T E = I$, so $E^{-1} = E^T$ and $E E^T = I$. (2) Then $\Sigma = \Sigma E E^T = [\Sigma \mathbf{e}_1 \cdots \Sigma \mathbf{e}_D] E^T = [\lambda_1 \mathbf{e}_1 \cdots \lambda_D \mathbf{e}_D] E^T = E \Lambda E^T$.

5.2 The Theoretical Number: $\text{rank}(\Sigma)$

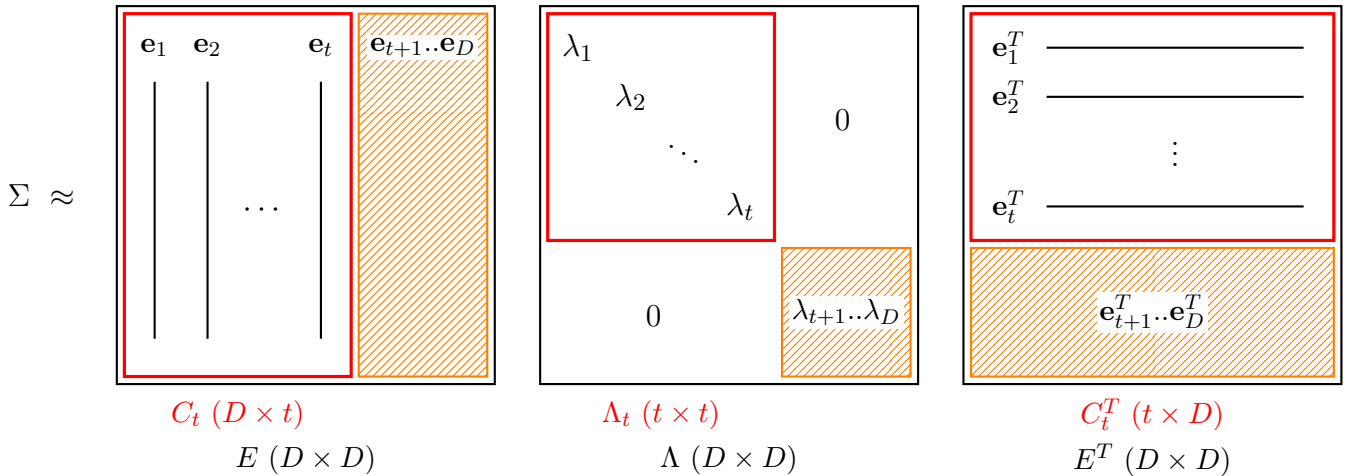
Equation 17 says the data's variation is fully described by D perpendicular directions, the j -th carrying variance λ_j . If some $\lambda_j = 0$, the data has *zero* spread along $\mathbf{e}_j$: every centered datapoint lies exactly in the subspace spanned by the other directions, and a new column along $\mathbf{e}_j$ would be constant (pure redundancy). So

$$t_{\text{theory}} = \#\{j : \lambda_j > 0\} = \text{rank}(\Sigma) \leq \min(D, N - 1), \quad (18)$$

the number of genuinely independent linear aggregates in the data. ($N - 1$ because the N centered rows sum to zero, so they span at most $N - 1$ dimensions.) Typical reasons for $\text{rank}(\Sigma) < D$: a feature that is an exact copy or linear combination of others, or fewer datapoints than features.

5.3 Keeping t Directions = a Rank- t Approximation of Σ

Write the spectral decomposition (17) as blocks: the first t eigenvectors and eigenvalues, which we keep, and the remaining $D - t$, which we ignore.



Red: kept (top t eigenvectors and eigenvalues). Orange, hatched: ignored. Setting $\lambda_{t+1}, \dots, \lambda_D$ to 0 in the middle matrix is the same as dropping the hatched blocks altogether.

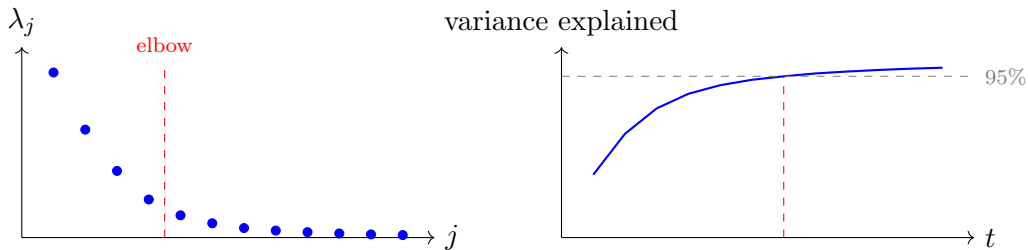
Keeping only the red blocks gives

$$\Sigma_t = C_t \Lambda_t C_t^T = \sum_{j=1}^t \lambda_j \mathbf{e}_j \mathbf{e}_j^T, \quad \Sigma - \Sigma_t = \sum_{j=t+1}^D \lambda_j \mathbf{e}_j \mathbf{e}_j^T. \quad (19)$$

- Σ_t is still $D \times D$, but it is built from only t vectors, so it has **rank at most t** (rank-deficient whenever $t < D$). Using the first t PCA dimensions is exactly the same as approximating the covariance matrix by this rank- t matrix: the data is treated as if it had no spread outside the span of $\mathbf{e}_1, \dots, \mathbf{e}_t$.
- **Information lost** is what was ignored: the $D - t$ dropped directions, each weighted by its eigenvalue. If $\lambda_{t+1}, \dots, \lambda_D$ are close to 0, then $\Sigma_t \approx \Sigma$ is a very good approximation; if they are exactly 0 (i.e. $t = \text{rank}(\Sigma)$), then $\Sigma_t = \Sigma$ and nothing is lost.
- So choosing t means: do not cut off any *large* eigenvalue.

5.4 Choosing t in Practice

- **Total variance is preserved:** $\sum_j \lambda_j = \text{trace}(\Sigma) = \sum_j \Sigma_{jj}$, the sum of the original features' variances. Rotating to the eigenvector coordinates redistributes the variance; it neither creates nor destroys it.
- **What you lose:** keeping the top t directions, the reconstruction error (§3, View 1) is exactly $N \sum_{j>t} \lambda_j$ — the variance in the dropped directions.
- **Variance explained** by the top t : $\sum_{j \leq t} \lambda_j / \sum_j \lambda_j$. Plot eigenvalues (and the cumulative fraction) in decreasing order; pick t at the “elbow,” or at a threshold such as 90% or 95%:



- **The task decides.** When PCA feeds a classifier, t is a hyperparameter like any other: choose it on a validation set (HW3 does exactly this for Kernel PCA). Small trailing eigenvalues are often noise, so dropping them can *improve* the classifier, not just speed it up. For visualization, $t = 2$ or 3 regardless.

6 The Algorithm

Stack the top t eigenvectors as columns: $C_t = [\mathbf{e}_1 \mathbf{e}_2 \dots \mathbf{e}_t]$ ($D \times t$). The new representation of a datapoint is its t projections, and the reconstruction maps back:

$$\mathbf{z} = (\mathbf{x} - \mu) C_t = [(\mathbf{x} - \mu)\mathbf{e}_1, \dots, (\mathbf{x} - \mu)\mathbf{e}_t] \quad (1 \times t), \quad \hat{\mathbf{x}} = \mu + \mathbf{z} C_t^T \quad (1 \times D). \quad (20)$$

Algorithm: PCA

1. **Fit** (training data only): $\mu = \text{mean of the training rows}$; $\tilde{X} = X_{train} - \mu$; $\Sigma = \frac{1}{N} \tilde{X}^T \tilde{X}$.
 2. Eigendecompose Σ ; sort eigenpairs by decreasing λ ; keep $C_t = [\mathbf{e}_1 \cdots \mathbf{e}_t]$.
 3. **Transform** any data (train, validation, test) with the *stored* μ and C_t : $Z = (X - \mu) C_t$.
-

7 Worked Example

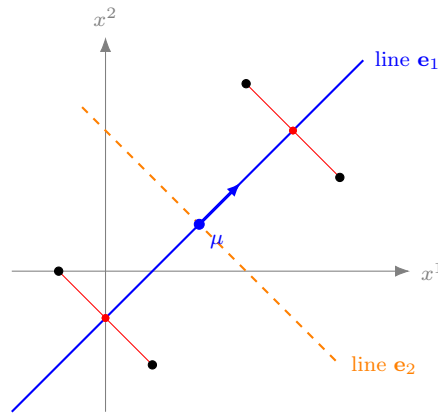
Four points in 2-D: $(5, 2)$, $(3, 4)$, $(1, -2)$, $(-1, 0)$. Mean $\mu = (2, 1)$; centered rows $(3, 1)$, $(1, 3)$, $(-1, -3)$, $(-3, -1)$.

$$\Sigma = \frac{1}{4} \left(\begin{bmatrix} 9 & 3 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 3 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 3 & 9 \end{bmatrix} + \begin{bmatrix} 9 & 3 \\ 3 & 1 \end{bmatrix} \right) = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}. \quad (21)$$

Eigenpairs (check by multiplying): $\Sigma(1, 1)^T = (8, 8)^T$, $\Sigma(1, -1)^T = (2, -2)^T$, so

$$\lambda_1 = 8, \mathbf{e}_1 = \frac{1}{\sqrt{2}}(1, 1)^T; \quad \lambda_2 = 2, \mathbf{e}_2 = \frac{1}{\sqrt{2}}(1, -1)^T. \quad (22)$$

- New coordinates on $\mathbf{e}_1$: $a_k = (\mathbf{x}_k - \mu)\mathbf{e}_1 = \frac{4}{\sqrt{2}}, \frac{4}{\sqrt{2}}, -\frac{4}{\sqrt{2}}, -\frac{4}{\sqrt{2}}$, variance $\frac{1}{4} \cdot 4 \cdot 8 = 8 = \lambda_1$. On $\mathbf{e}_2$: $\pm\sqrt{2}$, variance $2 = \lambda_2$. ✓
- Total variance $\Sigma_{11} + \Sigma_{22} = 10 = \lambda_1 + \lambda_2$; $t = 1$ explains 80%.
- Reconstruction with $t = 1$: $\hat{\mathbf{x}}_1 = \mu + a_1 \mathbf{e}_1^T = (2, 1) + \frac{4}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}}(1, 1) = (4, 3)$, versus the original $(5, 2)$: squared error 2. Every point has error 2, total $8 = N\lambda_2$. ✓



Black: data. Red: projections onto the first principal direction (the $t = 1$ reconstruction) and the errors.
Dashed: the second direction, perpendicular to the first.

Key point: Use this example as a unit test for your HW3 implementation: on these four points your Σ , eigenvalues, and projections should match the numbers above exactly (eigenvector signs may flip — an eigenvector times -1 is still an eigenvector).

A redundant feature. Append a third feature that is an exact copy of the first: centered rows $(3, 1, 3), (1, 3, 1), (-1, -3, -1), (-3, -1, -3)$, so

$$\Sigma = \begin{bmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{bmatrix}, \quad \Sigma(1, 0, -1)^T = (0, 0, 0)^T.$$

The direction $\frac{1}{\sqrt{2}}(1, 0, -1)$ (“feature 1 minus its copy”) has eigenvalue 0: the data has no spread there, $\text{rank}(\Sigma) = 2 = t_{\text{theory}}$, and PCA discards the redundancy automatically. The other two eigenvalues are $(15 \pm \sqrt{97})/2 \approx 12.4, 2.6$ — note the top one now captures 83% instead of 80%: the duplicated feature counts twice, pulling the first direction toward it. Both effects matter for HW3’s polluted Spambase.

8 Practical Notes

- **Standardize first (usually).** Σ depends on the features’ units: a feature measured in large numbers has a large variance and will dominate $\mathbf{e}_1$ purely because of scale. Unless the features already share a scale, z-score each feature (using training statistics) before PCA — this is PCA on the *correlation* matrix.
- **Computing it:** `np.linalg.eigh(Sigma)` is the right call for a symmetric matrix — it returns eigenvalues in *increasing* order, so reverse them. Equivalently use the SVD of the centered data, $\tilde{X} = USV^T$: then $\Sigma = \frac{1}{N}VS^2V^T$, so the columns of V are the principal directions and $\lambda_j = s_j^2/N$. SVD is more numerically stable and avoids forming the $D \times D$ matrix when D is large.
- **Fit on train only**, then apply the same μ , scaling, and C_t to test — exactly like normalization (HW1/HW2).
- **PCA is unsupervised:** it never looks at the labels. The highest-variance directions are not guaranteed to be the most *discriminative* ones — see LDA (§10.2).

8.1 A Second Look: Directions Are Combinations of Datapoints

From $\Sigma \mathbf{e} = \lambda \mathbf{e}$ with $\Sigma = \frac{1}{N} \tilde{X}^T \tilde{X}$ (and $\lambda > 0$):

$$\mathbf{e} = \frac{1}{N\lambda} \tilde{X}^T \tilde{X} \mathbf{e} = \tilde{X}^T \beta, \quad \beta = \frac{1}{N\lambda} \tilde{X} \mathbf{e} \quad (N \times 1), \quad (23)$$

i.e. $\mathbf{e} = \sum_k \beta_k \tilde{\mathbf{x}}_k^T$: every principal direction is a linear combination of the (centered) datapoints, with coefficients β_k . Then a projection $\tilde{\mathbf{x}} \mathbf{e} = \sum_k \beta_k \tilde{\mathbf{x}} \tilde{\mathbf{x}}_k^T$ needs only *dot products between datapoints*, never the features themselves.

Key point: *Forward pointer.* This observation — write the solution as a combination of datapoints, so that the algorithm only ever touches dot products $\mathbf{x}_i \mathbf{x}_j^T$ — is the door to the **Kernels** note. There, replacing those dot products with a nonlinear similarity turns PCA into **Kernel PCA** (HW3 Problem 4), and the same trick will work for regression, the perceptron, and SVMs.

9 Implementation Notes (HW3)

- **Problem 3** (polluted Spambase): (A) a classifier on the raw polluted features, compared against the same classifier on the clean 57 features; (B) library PCA (`sklearn.decomposition.PCA`) to ~ 100 components, then retrain; (C) your own PCA following §6. Fit scaler and PCA on the training split only.
- **Thinking about why PCA helps there:** recall what PCA does with a duplicated column (the redundant-feature example, §7) and with columns that carry little variance; then ask how many inputs, and how correlated, the classifier sees before and after PCA. Standardize first, so that no column dominates just by its scale.
- **Check against the library:** your eigenvalues should match `pca.explained_variance_` (up to the $\frac{1}{N}$ vs. $\frac{1}{N-1}$ factor), and your directions `pca.components_` (rows there, columns here) up to sign. Downstream accuracy should be essentially identical.
- **Common bugs:** forgetting to center; using `eigh`'s increasing order as if it were decreasing; re-fitting PCA on the test set; taking rows of the eigenvector matrix instead of columns.

10 Optional: Other Feature Representations

10.1 Whitening

Dividing each PCA coordinate by its standard deviation, $z^j/\sqrt{\lambda_j}$, gives features with unit variance *and* zero correlation (covariance = I). Often a preprocessing step for other algorithms, including ICA below.

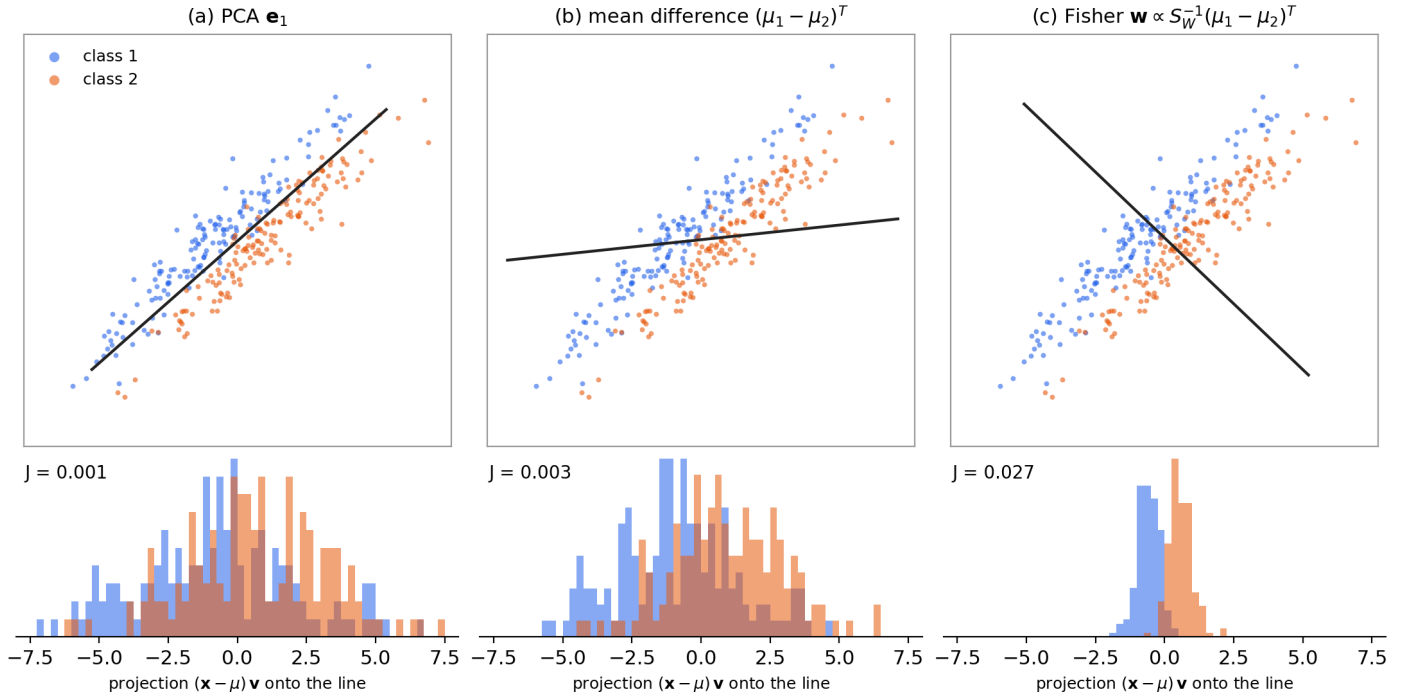
10.2 Fisher's Linear Discriminant (LDA)

PCA ignores labels. For two classes, LDA asks instead for the direction $\mathbf{w}$ along which the two class means are *far apart relative to the spread within each class*. With class means μ_1, μ_2 ($1 \times D$), projected means $m_c = \mu_c^T \mathbf{w}$, and projected within-class scatter $s_c^2 = \sum_{k \in c} (\mathbf{x}_k^T \mathbf{w} - m_c)^2 = \mathbf{w}^T S_c \mathbf{w}$ where $S_c = \sum_{k \in c} (\mathbf{x}_k - \mu_c)(\mathbf{x}_k - \mu_c)^T$:

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}, \quad S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T, \quad S_W = S_1 + S_2. \quad (24)$$

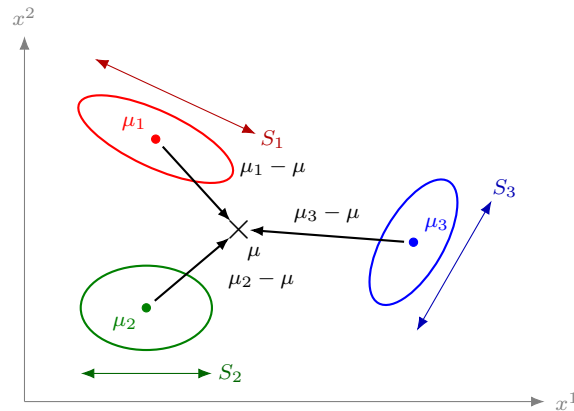
J does not change if $\mathbf{w}$ is rescaled, so fix $\mathbf{w}^T S_W \mathbf{w} = 1$ and use a Lagrangian once more: $L = \mathbf{w}^T S_B \mathbf{w} - \lambda(\mathbf{w}^T S_W \mathbf{w} - 1)$, $\partial L / \partial \mathbf{w} = 0 \Rightarrow S_B \mathbf{w} = \lambda S_W \mathbf{w}$ (a *generalized eigenproblem*). Since $S_B \mathbf{w} = (\mu_1 - \mu_2)^T [(\mu_1 - \mu_2) \mathbf{w}]$ always points along $(\mu_1 - \mu_2)^T$,

$$\mathbf{w} \propto S_W^{-1} (\mu_1 - \mu_2)^T. \quad (25)$$



Two elongated classes, projected onto three directions (top: the direction as a line through μ ; bottom: histograms of the projections, same scale in all three). (a) PCA ignores the labels and picks the long direction: the classes overlap. (b) Joining the class means is not enough either: along that line each class is still widely spread. (c) Fisher's $\mathbf{w}$ also accounts for the spread within each class and separates them; it has the largest J .

For C classes, project onto the top $C - 1$ eigenvectors of $S_W^{-1}S_B$ with $S_W = \sum_c S_c$ (spread inside each class) and $S_B = \sum_c N_c(\mu_c - \mu)^T(\mu_c - \mu)$ (spread of the class means around the overall mean μ):



Three classes. S_W collects the spreads S_c inside each class (double arrows); S_B collects the offsets $\mu_c - \mu$ of the class means from the overall mean (arrows). LDA looks for directions where the means are far apart compared with the spread inside the classes.

Practical: in high dimensions S_W can be singular. In `sklearn` it is regularized by `LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')`. HW3 has an optional “LDA instead of PCA” problem.

10.3 Independent Component Analysis (ICA) — Overview

The cocktail party. Three people talk at once; three microphones each record a different mixture. Can we recover the three voices? Model: each observed row is a linear mixture of unknown *independent* sources, $X = S A$ (S : $N \times D$ sources, A : unknown $D \times D$ mixing matrix). ICA estimates an unmixing matrix $W \approx A^{-1}$ so that the columns of XW are as statistically independent as possible.

- **PCA vs. ICA:** PCA makes the new columns *uncorrelated* (§4); ICA makes them *independent*, a much stronger requirement (no relationship of any kind, not just no linear one). For Gaussian data the two coincide — which is exactly why ICA needs *non-Gaussian* sources and cannot separate Gaussian ones.
- **How:** by the central limit theorem a mixture of independent signals looks more Gaussian than the signals themselves, so ICA searches for directions that are as *non-Gaussian* as possible (measured by kurtosis or negentropy). FastICA does this with a fixed-point iteration, usually after whitening.
- **Ambiguities:** sources come back in arbitrary order, sign, and scale; ICA components are not ranked by variance the way PCA's are.
- **Library:** `sklearn.decomposition.FastICA`. HW3 optional Problem 8 compares PCA, ICA, Kernel PCA, and t-SNE, including a cocktail-party-style unmixing task.

10.4 Feature Aggregates Without Learning the Directions

PCA *learns* its linear aggregates from the data. Two families of fixed, hand-designed aggregates:

- **Haar features** (images): differences of pixel sums over adjacent rectangles (left vs. right half, top vs. bottom half), computed in constant time per rectangle via an integral image. Used in HW1's MNIST features and in the Viola–Jones face detector (see the Haar_features folder in the course notes).
- **Random projections:** $Z = XR$ with a random $D \times t$ matrix R (e.g. Gaussian entries). By the Johnson–Lindenstrauss lemma, pairwise distances are approximately preserved once t grows like $\log N$, regardless of D — a cheap alternative to PCA when D is huge.

10.5 Feature Selection and Feature Analysis

Instead of building new columns, *keep a subset* of the original ones. Filter, wrapper, and embedded (Lasso) methods are covered in the **Logistic Regression Loss** note (LOGREG_LOSS); boosting-based feature analysis (rank features by their share of the ensemble's margin) is in the boosting-margin notes linked from the schedule. The trade-off versus PCA: selected features stay interpretable (“word frequency of *free*”), while PCA components are mixtures of all features.

10.6 t-SNE

A nonlinear method built to preserve *local neighborhoods* for 2-D/3-D visualization, rather than global variance. Not a linear aggregate and with no transform for new points. See the paper and slides in 5_features_dimensions/tSNE/; HW3 optional Problem 7 runs it (library, then from scratch).