Lecture 12: PCA, High Dimensionality, and Midterm Review

CS109A: Introduction to Data Science

Harvard University

Course:	CS109A: Introduction to Data Science
Lecture:	Lecture 12
Instructors:	Pavlos Protopapas, Kevin Rader, Chris Gumb
Topics:	Bayesian simulation (MCMC), Big Data challenges, Curse of Dimensionality, Principal Components Analysis (PCA), Hypothesis Testing review, Permutation Tests

Contents

Lecture Overview

This lecture wraps up the material before the midterm exam. We cover three major topics:

1. **Bayesian Simulation (MCMC):** How to work with complex posterior distributions when closed-form solutions don't exist
2. **High Dimensionality and PCA:** Understanding the "curse of dimensionality" and using Principal Components Analysis to handle many predictors
3. **Midterm Review:** Key concepts in hypothesis testing and permutation tests

Important: All material through this lecture (Lecture 12) is covered on the midterm. Classification modeling (starting next week) is NOT on the midterm.

1.1 Midterm Exam Information

- **When:** Next week during section time
- **In-class portion:**
  - 75 minutes
  - Approximately 2.2x the length of the quiz

  - Multiple choice and short answer/fill-in-the-blank
  - **Closed book**, but 2 cheat sheets allowed (front and back)
- **Take-home coding portion:**
  - Released after the in-class exam
  - 24-hour window to start, 2-hour time limit once started
  - No AI/LLM allowed; notes permitted
  - Best practice: homework problems and section material

2 Bayesian Simulation: Working with Complex Posteriors

2.1 Why Simulation?

In Bayesian inference, we combine a **prior distribution** with a **likelihood** to get a **posterior distribution**. When using conjugate priors, this posterior has a nice closed-form solution (e.g., Normal-Normal gives Normal posterior).

But in realistic models, especially linear regression with multiple parameters $(\beta_0, \beta_1, \ldots, \beta_p, \sigma^2)$, the **joint posterior distribution** becomes:

- **High-dimensional:** Many parameters to estimate simultaneously
- **Complex:** No simple mathematical form
- **Hard to integrate:** Can't compute means, variances, or credible intervals analytically

Key Summary

The Solution: Simulation
Instead of solving for the posterior mathematically, we **draw thousands of samples** from it. With enough samples, we can approximate any property of the distribution:

- **Posterior mean:** Sample average
- **Credible interval:** Sample percentiles (just like bootstrap!)
- **Posterior mode:** Requires kernel density estimation ("bump hunting")

2.2 MCMC: Markov Chain Monte Carlo

Definition: MCMC

Markov Chain Monte Carlo is a family of algorithms for sampling from probability distributions when direct sampling is difficult. The key insight: we don't need to know the entire distribution—we only need to be able to evaluate the **relative height** (probability density) at any point.

2.2.1 The Normal-Gamma Model

For Bayesian linear regression with conjugate priors:

- $\beta|\sigma^2, X \sim$ Normal (conditional on variance)
- $1/\sigma^2|X \sim$ Gamma

Sampling procedure:

1. First sample σ^2 from its Gamma distribution
2. Then sample β from its Normal distribution (conditional on the sampled σ^2)
3. Repeat to get pairs (σ^2, β) from the joint posterior

2.2.2 Gibbs Sampling

Definition: Gibbs Sampling

When we can't sample from the joint distribution directly, but we CAN sample from each parameter's **conditional distribution** (given the other parameters), we use Gibbs sampling:

1. Initialize all parameters: $\theta_1^{(0)}, \theta_2^{(0)}, \theta_3^{(0)}$
2. For iteration $t = 1, 2, \ldots$:
   - Sample $\theta_1^{(t)}$ from $f(\theta_1 | \theta_2^{(t-1)}, \theta_3^{(t-1)}, X)$
   - Sample $\theta_2^{(t)}$ from $f(\theta_2 | \theta_1^{(t)}, \theta_3^{(t-1)}, X)$
   - Sample $\theta_3^{(t)}$ from $f(\theta_3 | \theta_1^{(t)}, \theta_2^{(t)}, X)$
3. After a "burn-in" period, keep the samples

2.2.3 Metropolis-Hastings Algorithm

Example: The Blindfolded Mountain Climber

Imagine the posterior distribution as a **mountain range**. You're blindfolded but can measure your current altitude (probability density).

Algorithm:

1. Start at some point x on the mountain
2. Propose a random step to point x^*
3. **Decide whether to move:**
   - If x^* is **higher** (higher probability): Always move there
   - If x^* is **lower**: Move with probability $R = f(x^*)/f(x)$
     - Gentle downhill ($R = 0.8$): 80% chance to move
     - Steep cliff ($R = 0.01$): Only 1% chance to move
4. Repeat thousands of times

Result: The climber spends more time at high-altitude (high-probability) locations. The path traced becomes a sample from the posterior distribution!

Important Note

Burn-in Period

The first several thousand samples depend heavily on where you started. These are discarded ("burned") before using the remaining samples for inference.

3 Big Data and High Dimensionality

3.1 What is "Big Data"?

When we talk about "big data," we need to distinguish between two very different situations:

Big N	Many observations (rows)
Big P	Many predictors (columns)

These create **different problems** and require **different solutions**.

3.2 When N is Large

Problems:

- **Computational cost:** Training becomes slow. Simple operations like calculating means take time with billions of observations.
- **Bias doesn't disappear:** If your data collection is biased, more data can actually make results **worse**, not better. ("Garbage in, garbage out" at scale)

Solutions:

- **Subsampling:** Randomly select 10% or 1% of your data for training. With millions of observations, even a small fraction contains enough information.

Interesting observation: When N is extremely large (millions+), statistical inference becomes less meaningful:

- Standard errors shrink to nearly zero
- Confidence intervals become point estimates
- **Everything becomes "statistically significant"** ($p < 0.05$) even for trivially small effects

3.3 When P is Large: High Dimensionality

This is the more challenging situation. When the number of predictors P approaches or exceeds N:

Problems:

- **Overfitting:** Too many degrees of freedom—the model memorizes noise
- **Multicollinearity:** High correlation among predictors becomes almost certain
- **Matrix inversion fails:** Can't compute $(X^T X)^{-1}$ in OLS
- **Curse of dimensionality:** Fundamental geometric problem

Critical: The Curse of Dimensionality

As dimensions increase, the **volume of space grows exponentially**, causing data to become increasingly **sparse**.

Geometric intuition: The sphere inside the cube

- In 2D: A circle inscribed in a square occupies $\pi/4 \approx 78.5\%$ of the area

- In 3D: A sphere inscribed in a cube occupies $\pi/6 \approx 52.4\%$ of the volume
- In 10D: A 10D-sphere in a 10D-cube occupies only $\approx 0.25\%$
- As $d \to \infty$: This ratio approaches 0

What this means:

- In high dimensions, data points are "far apart"
- The concept of "nearest neighbor" breaks down
- Any two random points are approximately equidistant
- Local estimation (like k-NN) becomes unreliable

Example: Distance in High Dimensions

Sample two points from independent Normal distributions:

- In 1 dimension: Average distance is relatively small
- In 10 dimensions: Points are farther apart on average
- In 1000 dimensions: **All points are approximately equally far from each other**

This makes it very hard to estimate smooth functions, leading to overfitting.

3.4 When Does High Dimensionality Occur?

- **Polynomial regression:** 100 predictors with degree 20 = 2000+ terms
- **Interaction terms:** P main effects yields $\binom{P}{2}$ two-way interactions
- **Genomics:** Tens of thousands of genetic markers, but only hundreds of patients
- **NLP/Text:** Dictionary of 10,000+ words, each becoming a feature
- **Image data:** Each pixel is a feature (e.g., 784 features for 28×28 images)

3.5 How sklearn Handles Perfect Collinearity

When you have perfect collinearity (e.g., two identical columns), sklearn doesn't crash—it **splits the coefficient**:

Example: sklearn's Behavior

Predicting house price from square footage:

- Single predictor: $\hat{\beta}_{\text{sqft}} = 588$
- Duplicate predictor (sqft1, sqft2 are identical): $\hat{\beta}_{\text{sqft1}} = 294$, $\hat{\beta}_{\text{sqft2}} = 294$

This is problematic for interpretation (you can't change one without the other), but **reasonable for prediction** (splits predictive power equally).

4 Principal Components Analysis (PCA)

PCA is a powerful technique for **dimensionality reduction**—transforming high-dimensional data into a lower-dimensional representation while preserving as much information as possible.

4.1 The Core Idea

Definition: Principal Components Analysis

PCA finds a **linear transformation** of your original predictors $X_1, X_2, \ldots, X_P$ into new variables $Z_1, Z_2, \ldots, Z_P$ such that:

- Z_1 (first principal component) captures the **maximum variance** in the data
- Z_2 is **orthogonal** to Z_1 and captures the maximum **remaining** variance
- Each subsequent Z_k is orthogonal to all previous components

The key insight: if predictors are correlated, the first few Z's can capture most of the "information" (variance) in all P original variables.

Example: Rotating the Axes

Imagine a 2D scatter plot of (X_1, X_2) that forms an elongated ellipse tilted at 45 degrees.

- **Original axes:** X_1 (horizontal) and X_2 (vertical) don't align with the data's natural structure
- **PCA axes:**
  - Z_1: Points along the "spine" of the ellipse (maximum spread)
  - Z_2: Perpendicular to Z_1 (remaining spread)

If the ellipse is very "thin," then Z_1 alone captures almost all the information. We've **reduced** from 2 dimensions to 1!

4.2 Mathematical Foundation

PCA is mathematically equivalent to finding the **eigenvectors** of the covariance matrix $X^T X$:

- **Eigenvectors** determine the **directions** of the principal components
- **Eigenvalues** determine the **importance** (variance explained) of each component

The eigenvector with the largest eigenvalue becomes PC1, the second largest becomes PC2, etc.

Key Summary

PCA is a Rotation

PCA performs a **linear transformation** (rotation) of your coordinate system to align with the directions of maximum variance in the data. The new coordinates are called principal components. Each principal component Z_k is a **linear combination** of all original variables:

$$Z_k = w_{k1}X_1 + w_{k2}X_2 + \cdots + w_{kP}X_P \tag{1}$$

The weights w_{ki} come from the eigenvectors and tell us how much each original variable contributes

to each component.

4.3 PCA in sklearn

```python
from sklearn.decomposition import PCA
import numpy as np

# Original data: n observations, p features
X = data[['sqft', 'beds', 'baths', 'lot_size', 'distance']]

# Fit PCA
pca = PCA()
pca.fit(X)

# Transform data to principal components
X_pca = pca.transform(X)

# Get the component weights (loadings)
print("Component 1 weights:", pca.components_[0])
# Output: [ 0.95  0.28  0.08  0.11  0.02]
# Interpretation: PC1 is mostly sqft and beds

# Variance explained by each component
print("Variance ratios:", pca.explained_variance_ratio_)
# Output: [0.91, 0.05, 0.02, 0.01, 0.01]
# Interpretation: PC1 alone captures 91% of variance
```

4.4 PCA for Visualization

One of the most common uses of PCA is **visualizing high-dimensional data** in 2D:

1. Run PCA on your P-dimensional data
2. Keep only PC1 and PC2
3. Plot each observation as a point with (Z_1, Z_2) coordinates
4. Color by class/category to see if groups separate

Example: Penguin Species Visualization

Data: 4 measurements (bill length, bill depth, flipper length, body mass) for 3 penguin species
Problem: Can't visualize 4D data directly
Solution:

- Run PCA on the 4 measurements
- Plot penguins using PC1 (x-axis) and PC2 (y-axis)
- Color by species

Result: The three species form distinct clusters in PC space, even though PCA never saw the species labels!

Example: Fashion MNIST

Data: 28×28 grayscale images of clothing items (784 pixels = 784 features)
Categories: T-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, ankle boots
PCA Visualization:

- Run PCA on the 784-dimensional pixel data
- Plot using PC1 and PC2
- Color by clothing category

Result: While there's overlap, items of the same category tend to cluster together. This suggests the pixel features contain information useful for classification.

Important Note

PCA is Unsupervised
PCA only looks at the predictor variables X. It has **no knowledge of the response** Y (class labels, outcomes, etc.).
If clusters separate well in PCA space, it's a good sign that X contains information predictive of Y. But PC1 is NOT guaranteed to be the "best predictor" of Y—it's just the direction of maximum variance in X.

4.5 PCA for Regression (PCR)

Principal Components Regression uses PCA as a **preprocessing step** to handle high dimensionality:

1. **Step 1: Run PCA** on all P predictors to get P principal components
2. **Step 2: Select M components** (where $M < P$)
3. **Step 3: Fit regression** using only the selected components:

$$Y = \beta_0 + \beta_1 Z_1 + \beta_2 Z_2 + \cdots + \beta_M Z_M + \epsilon \tag{2}$$

4.5.1 Choosing the Number of Components (M)

Method 1: Scree Plot / Elbow Method

- Plot variance explained by each component
- Look for an "elbow" where the curve levels off
- Keep components before the elbow

Method 2: Cumulative Variance Threshold

- Keep components until you capture X% of total variance
- Common thresholds: 90%, 95%
- Example: "53 components explain 90% of variance"

Method 3: Cross-Validation

- Treat M as a hyperparameter
- For each candidate M, compute cross-validation MSE

- Select M that minimizes validation error
- This is the most "performance-oriented" approach

4.6 Pros and Cons of PCA

Key Information

Advantages:

- **Reduces overfitting:** Fewer dimensions = less capacity to memorize noise
- **Eliminates multicollinearity:** Principal components are orthogonal by construction
- **Enables visualization:** Project any high-dimensional data to 2D/3D
- **Speeds up computation:** Fewer features = faster training

Important Note

Disadvantages:

- **Loss of interpretability:** $Z_1 = 0.5X_1 - 0.3X_2 + 0.2X_3 + \cdots$ is hard to explain to stakeholders
- **Ignores the response:** PC1 explains most variance in X, but might not predict Y well. The "important" information for Y could be in PC50!
- **No guaranteed improvement:** Prediction accuracy might not improve over using original features
- **Linear only:** PCA finds linear combinations; nonlinear relationships are missed

5 Midterm Review: Hypothesis Testing

5.1 The Framework

Definition: Hypothesis Testing

Hypothesis testing is a formal procedure to determine whether observed effects in data are "real" or could have occurred by random chance.

The Five Steps:

1. **State hypotheses:**
   - H_0 (null): "No effect" (e.g., $\beta_1 = 0$)
   - H_A (alternative): "There is an effect" (e.g., $\beta_1 \neq 0$)
2. **Choose test statistic:** A measure to evaluate the hypothesis (e.g., t-statistic)
3. **Calculate test statistic:** Compute it from your data
4. **Compute p-value:** How extreme is this statistic under H_0?
5. **Make a decision:**
   - If $p < \alpha$ (usually 0.05): Reject H_0
   - If $p \geq \alpha$: Fail to reject H_0

5.2 Understanding P-values

Critical: What is a P-value?

The p-value is the probability of observing a test statistic **as extreme or more extreme** than what we calculated, **assuming H_0 is true**.

Intuition:

- Small p-value (e.g., 0.01): "If there were truly no effect, there's only a 1% chance of seeing data this extreme. That's unlikely, so maybe H_0 is wrong."
- Large p-value (e.g., 0.40): "If there were no effect, we'd see data this extreme 40% of the time. Not surprising at all."

Mnemonic: "If the p-value is low, H_0 must go!"

5.3 T-Test for Regression Coefficients

For testing whether $\beta_1 = 0$ in simple linear regression:

$$t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)} = \frac{\hat{\beta}_1}{SE(\hat{\beta}_1)} \tag{3}$$

Under H_0, this follows a t-distribution with $n - 2$ degrees of freedom.

Example: Testing Housing Price Relationship

Question: Is there a real relationship between square footage and house price?

Data: 592 homes in Cambridge/Somerville

Results from statsmodels:

- $\hat{\beta}_1 = 0.589$ (price increases \$589 per additional sqft)
- $SE(\hat{\beta}_1) = 0.023$
- $t = 0.589/0.023 = 25.2$
- $p \approx 0$ (reported as 0.000)

Conclusion: With $p < 0.05$, we reject H_0. There is statistically significant evidence that square footage is associated with house price.

But wait—our assumptions (especially constant variance) are violated!

6 Permutation Tests

6.1 When Assumptions Fail

The t-test relies on assumptions:

- Linearity
- Independence
- **Normality of residuals**
- **Constant variance (homoscedasticity)**

When these are violated (especially for small samples), the p-value from a t-test may not be trustworthy.

6.2 The Permutation Test Idea

Definition: Permutation Test

A **permutation test** is a non-parametric method that simulates the null hypothesis (H_0: no relationship between X and Y) by **randomly shuffling** the Y values.
Key insight: If X and Y are truly unrelated, then it shouldn't matter which Y value is paired with which X value.

6.3 Permutation Test Procedure

1. **Calculate observed statistic:** Compute $\hat{\beta}_1$ from the original data (e.g., 0.589)
2. **Simulate H_0:** Randomly shuffle (permute) the Y values while keeping X fixed
3. **Calculate permuted statistic:** Fit regression on (X, Y_{shuffled}) and get $\hat{\beta}_{\text{perm}}$
4. **Repeat:** Do steps 2-3 many times (e.g., 1000 or 10000 iterations)
5. **Build null distribution:** The 1000 permuted $\hat{\beta}$ values represent what we'd see if H_0 were true
6. **Calculate p-value:**

$$p = \frac{\text{Number of permuted } |\hat{\beta}| \geq \text{observed } |\hat{\beta}|}{\text{Total permutations}} \tag{4}$$

Example: Permutation Test for Housing Data

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Original data
X = houses['sqft'].values.reshape(-1, 1)
y = houses['price'].values

# Observed coefficient
model = LinearRegression().fit(X, y)
observed_beta = model.coef_[0]  # 0.589

```

```python
# Permutation test
n_perms = 10000
perm_betas = []

for _ in range(n_perms):
    y_shuffled = np.random.permutation(y)  # Shuffle Y
    model_perm = LinearRegression().fit(X, y_shuffled)
    perm_betas.append(model_perm.coef_[0])

# P-value: proportion of permuted betas >= observed
p_value = np.mean(np.abs(perm_betas) >= np.abs(observed_beta))
print(f"Permutation p-value: {p_value}")
# Output: 0.0000 (none of 10000 permutations produced such extreme value)
```

6.4 Bootstrap vs. Permutation Tests

Key Summary

	Bootstrap	Permutation Test
Goal	Estimation	Hypothesis testing
Question	"How uncertain is my estimate?"	"Is the effect real?"
Output	Confidence interval	P-value
Assumes	H_A (observed data is representative)	H_0 (no relationship)
Resampling	With replacement (from (x_i, y_i) pairs)	Without replacement (shuffle Y only)

7 Key Concepts Summary

Key Summary

Bayesian Simulation (MCMC):

- When posterior distributions are complex, we **sample** from them instead of solving analytically
- MCMC algorithms (Gibbs, Metropolis-Hastings) generate samples by exploring the probability landscape
- Use sample statistics (mean, percentiles) to estimate posterior properties

High Dimensionality:

- Big N (many rows): Computational cost, but more data is generally good
- Big P (many columns): Overfitting, multicollinearity, curse of dimensionality
- Curse of dimensionality: Data becomes sparse; all points become "far apart"

PCA:

- Finds linear combinations of predictors that maximize variance
- PC1 captures most variance, PC2 captures most remaining variance, etc.
- Uses: Visualization (plot PC1 vs PC2), Regression (use top M components)
- Limitation: Ignores Y; loses interpretability

Hypothesis Testing:

- P-value = probability of seeing data this extreme if H_0 is true
- Reject H_0 if p-value < 0.05 (typically)
- T-test requires assumptions; permutation test is the non-parametric alternative

Bootstrap vs. Permutation:

- Bootstrap: Sample with replacement $\rightarrow$ Confidence intervals
- Permutation: Shuffle $Y \rightarrow$ P-values under H_0

8 Quick Reference: Key Formulas

Key Information

PCA Transformation:

$$Z_k = \sum_{j=1}^{P} w_{kj} X_j \quad \text{(linear combination)} \tag{5}$$

Variance Explained:

$$\text{Proportion} = \frac{\lambda_k}{\sum_{j=1}^{P} \lambda_j} \quad \text{(eigenvalue ratio)} \tag{6}$$

T-Statistic:

$$t = \frac{\hat{\beta} - \beta_0}{SE(\hat{\beta})} \sim t_{n-p-1} \tag{7}$$

Permutation P-value:

$$p = \frac{1}{B} \sum_{b=1}^{B} \mathbf{1}(|\hat{\beta}_{\text{perm}}^{(b)}| \geq |\hat{\beta}_{\text{obs}}|) \tag{8}$$

Confidence Interval (Bootstrap):

$$[\hat{\beta}_{2.5\%}, \hat{\beta}_{97.5\%}] \quad \text{(percentile method)} \tag{9}$$