CS109A: Introduction to Data Science
Lecture 01: What is Data Science?

Harvard University

Fall 2024

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 01: Course Introduction
- **Instructor:** Pavlos Protopapas, Kevin Rader, Chris Gumb
- **Objective:** Understand the definition, history, and process of data science; learn course policies and expectations

Key Summary

This document is a comprehensive guide to the first lecture of CS109A "Introduction to Data Science" at Harvard University. It covers the fundamental definition of data science, its historical development from ancient times to the modern era, the five-step data science process, and the three core components that make up this interdisciplinary field. Additionally, it provides detailed course logistics including grading policies, the critical attendance requirements, and important prerequisites. The lecture concludes with an introduction to web scraping as the first practical skill students will learn.

Contents

1 Course Overview

1.1 What This Course Is About

Welcome to CS109A, also known as AC209A, STAT 109A, and CSCI E-109A for Extension School students. This course is the **beginning of your journey** into data science and artificial intelligence—not the destination.

Warning

Setting Realistic Expectations

You will **not** become an expert in AI and data science by taking this course alone. Think of this course as laying the foundation. There are many more things to learn after this, including:

- CS109B: Advanced Topics in Data Science (Spring semester)
- AC215: Advanced Practical Data Science
- Machine Learning, NLP, and Deep Learning courses

1.2 The "Wax On, Wax Off" Philosophy

Professor Protopapas describes the teaching philosophy using a famous scene from the movie "The Karate Kid." In this movie, the teacher (Mr. Miyagi) makes his student repeatedly wax cars with specific motions—"wax on, wax off"—before teaching any actual karate. The student is frustrated, but later discovers that these motions became the foundation for perfect defensive moves.

Example:

The Karate Kid Analogy **The Modern Temptation:** Students often want to jump straight to running large language models (LLMs), building AI bots, and starting companies.

The Old-Fashioned Approach: This course insists on understanding the fundamentals first. You will:

- Train models and validate them repeatedly
- Understand *what* is happening inside models
- Learn *why* models work the way they do
- Avoid the "dot fit pandemic"—blindly calling `model.fit()` without understanding

1.3 Learning Roadmap

The course follows a logical progression through the data science workflow:

1. **Weeks 1-2: Data Collection & Exploration**
   - Web scraping and data wrangling
   - Exploratory Data Analysis (EDA)
   - Data visualization
2. **Weeks 3-5: Regression**
   - K-Nearest Neighbors (KNN) for regression

- Simple and multiple linear regression
- Model selection with cross-validation
- Regularization (Ridge, Lasso)

3. **Week 6: Bayesian Modeling** (New this year!)
   - Bayesian inference framework
   - Bayesian linear regression
4. **Weeks 7-9: Classification**
   - KNN for classification
   - Logistic regression
   - **Midterm exam** during this period
5. **Week 10: Data Issues**
   - Missing data (Missingness)
   - Causal inference
   - Bias and ethics in data science
6. **Weeks 11-14: Tree-Based Models**
   - Decision trees
   - Bagging and Random Forests
   - Boosting methods

2 What is Data Science?

2.1 The Simple Answer

At its core, **data science is the process of extracting meaningful insights and value from data**. But to truly understand what this means, let's look at it from multiple perspectives.

Definition:

Data Science **Data Science** is an interdisciplinary field that combines:

- **Mathematics & Statistics**: For modeling, hypothesis testing, and prediction
- **Computer Science & IT**: For data collection, storage, processing, and software development
- **Domain Knowledge**: Expert understanding of the specific field being studied (medicine, astronomy, finance, etc.)

2.2 A Historical Perspective: Four Stages of Understanding the World

To understand where data science fits in human history, consider how humanity has approached understanding the world:

2.2.1 Stage 1: Empirical Observation (Ancient Times)

Long ago, humans learned by direct observation and recording:

- **Counting stars**: Ancient astronomers looked at the night sky and counted stars in different regions
- **Recording crops**: Farmers tracked harvests to predict yields
- **The Antikythera mechanism**: An ancient Greek computer (found at the bottom of the sea in Greece) that calculated planetary orbits using gears

This was essentially **early statistics**—collecting data and making observations.

2.2.2 Stage 2: First Principles and Equations (Modern Science)

Scientists like Newton, Einstein, Maxwell, and Schrödinger developed **fundamental equations** that describe how the universe works:

- Newton's Laws: $F = ma$ (Force equals mass times acceleration)
- Einstein's equation: $E = mc^2$ (Energy equals mass times the speed of light squared)
- Maxwell's equations: Describe electromagnetism
- Navier-Stokes equations: Describe fluid dynamics
- Schrödinger's equation: Describes quantum mechanics

This represented a shift from just *observing* to *understanding from first principles*.

2.2.3 Stage 3: Computation (20th Century)

A problem emerged: many of these fundamental equations are **too complex to solve analytically**. Mathematicians couldn't provide closed-form solutions to many differential equations.

Solution: Use **computers** to numerically simulate and approximate solutions. This gave birth to computational science.

2.2.4 Stage 4: Data Science (Modern Era)

Key Information

The Key Insight of Data Science

Data science represents a paradigm shift: instead of requiring complete understanding of first principles (Stage 2), we can use **massive amounts of data (Stage 1)** combined with **powerful computing (Stage 3)** to approximate or predict how the world works.

In other words, data science sometimes "skips" the equation stage, focusing instead on patterns in data.

This can feel **unsettling for mathematically-oriented students**. Professor Protopapas acknowledges: "It bothers me too, but first step—let's learn how to do it, and then we'll learn how to do more than that."

2.3 The Three Pillars of Data Science

Data science exists at the intersection of three domains:

Table 1: *The Three Pillars of Data Science*

Pillar	Description
Math & Statistics	Probability, statistical inference, linear algebra, calculus—the mathematical foundation for modeling and prediction
Computer Science	Programming, algorithms, data structures, databases—the tools for handling and processing data
Domain Knowledge	Understanding of the specific field (astronomy, medicine, finance)—essential for asking the right questions

Warning

The Critical Importance of Domain Knowledge

Professor Protopapas shares a personal story: As an astronomer, he once gave a research problem to a computer scientist friend. A month later, the friend said "Everything is solved!" But it turned out they solved the **wrong problem** because they didn't understand the astronomical context.

Lesson: Data science is **not** like taking your car to a mechanic. You can't just hand over your data and say "Find something interesting." You must:

- Stay involved with the analysis
- Understand at least the basics of the domain
- Collaborate closely with domain experts

3 The Potential and Risks of Data Science

3.1 Amazing Applications

Data science and AI have tremendous potential for positive impact:

Positive Applications

1. **Medical Diagnosis**: Detecting malaria from blood smear images automatically
2. **Drug Discovery**: Using language models to discover new drug combinations
3. **Autonomous Vehicles**: Self-driving trucks for safe night shipping
4. **Generative AI**: Creating images from text prompts (e.g., "a Greek-American professor with glasses and a beard")

3.2 Critical Risks and Ethical Concerns

Serious Risks and Biases

1. **Gender Bias**: Hiring algorithms that favor male candidates for engineering roles because they were trained on historical data that reflected past discrimination
2. **Racial Bias**: Recidivism prediction models (like COMPAS used in US courts) that unfairly predict higher risk for people of color
3. **Misinformation**: AI-generated content that's indistinguishable from reality

Important:

Being a Critical Thinker AI models are trained on data that contains human biases. As Harvard students, you must:

- Question the data: Where did it come from? Who collected it? What biases might exist?
- Question the model: Is it fair? Does it work equally well for all groups?
- Question the application: Should this model be used at all? What are the consequences?

Don't just accept AI results because a computer produced them. **Be critical thinkers.**

3.3 Career Advice for Uncertain Times

Professor Protopapas acknowledges that students today face unprecedented uncertainty:

- AI might replace jobs
- It's hard to distinguish real from fake information
- Career paths that existed yesterday might disappear tomorrow

Key Information

Simple Advice

"Don't try to over-optimize for every trend. Things change so fast. Instead: **Learn things very**

well, become good at them, and lead. Don't ignore trends—they are there—but don't let them control your life either."
— Professor Protopapas

4 The Five-Step Data Science Process

Data science projects typically follow a cyclical process with five key stages:

4.1 Step 1: Ask an Interesting Question

This is the **most important and first step**. Many projects fail because they start with "Here's some data, find something interesting" rather than a clear hypothesis.

Definition:

Good Questions in Data Science Good data science questions are:

- **Specific**: "Can we predict hospital readmission rates within 30 days?" rather than "Tell me something about hospitals"
- **Measurable**: There must be data that can address the question
- **Actionable**: The answer should lead to meaningful action
- **Scientific**: Based on hypotheses that can be tested

Key questions to ask yourself:

- What are we trying to predict or estimate?
- If we had all the data in the world, what would we do with it?

4.2 Step 2: Get the Data

Once you have a question, you need data to answer it.

Warning

Data Collection Considerations

- **How was the data sampled?** Random? Convenience? This affects what conclusions you can draw
- **What data is relevant?** More data isn't always better—irrelevant data adds noise
- **Are there privacy concerns?** Just because data is "available" doesn't mean it's ethical or legal to use
- **What is the license?** Always read the terms of service and data licenses

4.3 Step 3: Explore the Data (EDA)

Exploratory Data Analysis (EDA) is often undervalued but critically important. Professor Protopapas emphasizes: "Trust me on this—I'm old enough to have seen so many attempts to do modeling without data exploration. You're wasting your time."

Why EDA matters:

- Sometimes the answer is immediately obvious from visualization
- Sometimes a simple filter or rule works better than a complex model
- Sometimes the data is garbage, and no model can help

Key EDA questions:

- Have you plotted the data?
- Are there outliers or anomalies?
- Are there obvious patterns?
- Is there missing data?

4.4 Step 4: Model the Data

This is "where the fun is"—building, fitting, and validating models. The course will spend significant time on this step, covering:

1. **Build**: Choose an appropriate model type
2. **Fit**: Train the model on your data
3. **Validate**: Test the model on held-out data

4.5 Step 5: Communicate/Visualize the Results

Data science is an **interdisciplinary field**, which means you'll often communicate with people who don't know your technical terminology.

Key Information

The Art of Storytelling

Professor Protopapas shares: "I love telling stories—for me it's natural." But for many data scientists, this is the hardest part.

A student submitted a project proposal with **three jargon terms in the first title**. The response: "Rewrite."

Key questions for communication:

- What did we learn?
- Does it make sense?
- Can we tell the story effectively to non-experts?

5 Why Choose Data Science?

5.1 It's Fun

If you're in this class, you probably enjoy problem-solving. Data science is fundamentally about solving problems with data.

5.2 It's at the Cutting Edge

You'll work with the latest technologies and methods.

5.3 Career Prospects

Table 2: *Data Science Career Statistics*

Metric	**Value**
Average Salary	High (varies by location and experience)
Job Satisfaction	Among the highest in tech careers
Job Availability	Consistently ranked among "Best Jobs in America"

5.4 It's Accessible

The barrier to entry is lower than many technical fields. With dedication, you can learn the skills needed to start working in data science.

Key Summary

Professor Protopapas's Life Philosophy

"In principle, if you learn these things, you get decent jobs and you enjoy it. I think that's the secret of life—doing something you like and not lacking money. I do believe lack of money brings unhappiness, but money may not bring happiness. So at least you eliminate that."

6 Course Logistics and Policies

6.1 The Teaching Team

6.1.1 Pavlos Protopapas

- Scientific Director for Data Science and CSC programs
- Research focus: Astronomy + Machine Learning + AI + Statistics (Lab: Stellar DNN)
- Fun facts: Certified chef (trained at Le Cordon Bleu), classical music enthusiast, self-proclaimed "worst soldier in NATO" during Greek army service

6.1.2 Kevin Rader

- Senior Preceptor in the Statistics Department, Associate DUS
- Research focus: Medicine and sports analytics
- Fun facts: Philadelphia Eagles superfan ("Go Birds!"), passionate about growing and cooking food, first-time girls' soccer coach (currently 0-1)

6.1.3 Chris Gumb

- Preceptor in Computer Science
- Helps coordinate the teaching staff and approximately 30 Teaching Fellows (TFs)
- Fun fact: Big movie fan (Homework 1 involves movie theater data)

6.2 Two Perspectives on Data Science

The course intentionally brings together two viewpoints:

Professor Protopapas: The CS/ML Perspective

Focus on:

- Machine learning and optimization
- Getting better models (higher R^2, lower MSE)
- Improving prediction accuracy

Professor Rader: The Statistical Perspective

Focus on:

- Understanding **relationships** between variables
- Quantifying **uncertainty** in predictions
- Examining whether relationships differ for different groups
- Considering **data issues, biases, and ethics**

"I don't care how accurate your model is—it could still be trash if you don't understand the relationships and uncertainty."

6.3 Grading Components

Table 3: *CS109A Grading Breakdown*

Component	Weight	Details
Homework	30%	HW0 (1%) + HW1-5 (29%). Pair work recommended for HW1-5.
Section Quizzes	10%	Two 30-minute quizzes during section (5% each)
Midterm	18%	Conceptual portion in section + take-home coding portion
Final Exam	22%	3-hour seated exam with conceptual and coding portions
Project	20%	Group project (3-5 students), open-ended data science project

6.4 The Critical Attendance Policy

Important:

Attendance Directly Affects Your Maximum Possible Grade Attendance in CS109A is **required** and directly impacts your grade ceiling:

Minimum Attendance	Maximum Possible Grade
$\geq$ 66% (two-thirds)	A
$\geq$ 50% (one-half)	A-
$\geq$ 33% (one-third)	B+
$<$ 33%	B or below

Important: Meeting the attendance threshold doesn't *guarantee* that grade—it only *qualifies* you for it. You still need to earn the grade through your work.

Flexibility: 66% means you can miss one-third of classes. You could attend all lectures and skip most sections (except for quizzes/midterm), or attend all sections and skip half of lectures.

6.5 Late Days

- **Earning Late Days**: For every **4 sessions** you attend (lecture or section), you earn **1 late day**
- **Using Late Days**: Maximum of **2 late days** per assignment
- **DCE Students**: Automatically receive **4 late days** (attendance can't be tracked remotely)
- **48-hour limit**: After the deadline plus your late days, assignments cannot be submitted (grading must begin)

6.6 Prerequisites

Table 4: *CS109A Prerequisites*

Area	Requirements
Python Programming	**Critical**. If you have never programmed before, this course will be very challenging. CS50 or equivalent experience required.
Calculus	Basic calculus (Math 1B equivalent). Occasional linear algebra and multivariable calculus concepts appear but won't be tested heavily.
Statistics/Probability	Stat 104 (data-focused) is ideal. Stat 110 (theory-focused) is good for probability but may lack practical data experience.

Warning

Self-Assessment with Homework Zero

HW0 is designed to test whether you meet the prerequisites.

- If you struggle in **all three areas** (coding, math, stats): seriously consider taking the course next year
- If you struggle in **one area**: you can probably catch up with help from TFs
- **Coding experience is the most critical**: Math and stats gaps can be filled, but lack of coding experience is a "little bit more problematic"

6.7 Course Platforms

- **Ed (Edstem)**: Lecture slides, section materials, announcements, **discussion forum** (primary Q&A platform)
- **Canvas**: Video recordings, assignment submissions, official schedules, **grades**

6.8 Getting Help

In order of preference:

1. **Ed Discussion Forum**: Fastest response; classmates and TFs can help
2. **Office Hours**: Best for in-depth conceptual help or assignment debugging
3. **Course Helpline Email**: Administrative questions (e.g., section changes)
4. **Direct Email to Professors**: Personal or sensitive matters only

7 Introduction to Web Scraping

The first practical skill you'll learn in this course is **web scraping**—the automated extraction of data from websites.

7.1 What is Web Scraping?

Web scraping is a technique that allows you to programmatically collect data from websites. Instead of manually copying information, you write code that:

1. Fetches the HTML source code of a webpage
2. Parses the HTML to find specific elements
3. Extracts the data you need
4. Stores it in a structured format (like a spreadsheet or database)

7.2 Key Libraries

- **requests**: Makes HTTP requests to fetch webpage content
- **BeautifulSoup**: Parses HTML and makes it easy to navigate and search
- **pandas**: Organizes extracted data into DataFrames for analysis
- **matplotlib**: Visualizes the collected data

7.3 Ethical Considerations

Warning

Before You Scrape

Not all websites allow scraping. Always check:

- **robots.txt**: Visit `website.com/robots.txt` to see what's allowed/disallowed
- **Terms of Service**: Read the website's ToS for data collection policies
- **Rate Limiting**: Don't overwhelm servers with too many requests too quickly
- **Personal Data**: Be especially careful with data that might contain personal information

Just because data is "publicly available" doesn't mean it's okay to scrape and use it.

7.4 Basic HTML Concepts

Websites are written in **HTML (HyperText Markup Language)**. Understanding basic HTML helps you scrape effectively:

- **Elements**: Everything in HTML is organized into elements, marked by tags
- **Tags**: Define the type of content: `<h1>` (heading), `<p>` (paragraph), `<a>` (link), `<div>` (division/-container)
- **Attributes**: Provide additional information: `href` (link destination), `class` (styling identifier), `id` (unique identifier)

Example:

Using Browser Developer Tools The easiest way to understand a webpage's structure:

1. Right-click on the element you're interested in
2. Select "Inspect" or "Inspect Element"
3. The Developer Tools panel opens, highlighting the HTML for that element
4. Use the picker tool (arrow icon) to click on different elements and see their HTML

This is essential for figuring out which tags and classes to target in your scraping code.

7.5 Step-by-Step Web Scraping Example

7.5.1 Step 1: Fetch the Webpage

```python
import requests

# URL of the webpage we want to scrape
url = "https://www.nobelprize.org/all-nobel-prizes/"

# Send an HTTP GET request
response = requests.get(url)

# Check if request was successful (status code 200 = OK)
print(f"Status Code: {response.status_code}")

# Get the HTML content as text
html_text = response.text
print(html_text[:200])  # Print first 200 characters
```

Listing 1: Fetching HTML with requests

Status Codes to Know:

- **200**: Success (OK)
- **404**: Page not found
- **403**: Forbidden (you're not allowed to access)
- **429**: Too many requests (you're being rate-limited)

7.5.2 Step 2: Parse the HTML

```python
from bs4 import BeautifulSoup

# Create a BeautifulSoup object
soup = BeautifulSoup(html_text, 'html.parser')

# Find the page title
page_title = soup.title.get_text()
print(f"Page Title: {page_title}")

```

```
# Find all elements with a specific class using CSS selectors
prize_blocks = soup.select('div.card-prize')
print(f"Found {len(prize_blocks)} prize blocks")

# Get text from the first block
first_block = prize_blocks[0]
block_text = first_block.get_text().strip()
print(block_text)
```

Listing 2: Parsing HTML with BeautifulSoup

7.5.3 Step 3: Extract and Structure the Data

```
import re  # Regular expressions library

# Helper functions using lambda
get_title = lambda block: block.select_one('h3').get_text().strip()
get_year = lambda block: re.search(r'(\d{4})',
                          block.select_one('h3').get_text()).group(1)
get_description = lambda block: block.select_one('blockquote').get_text().
    strip()

# List to store our data
nobel_data = []

# Loop through all prize blocks
for block in prize_blocks:
    try:
        nobel_data.append({
            'title': get_title(block),
            'year': int(get_year(block)),
            'description': get_description(block)
        })
    except Exception as e:
        print(f"Error extracting data: {e}")

# Example analysis: Find unique prize categories
unique_titles = set(item['title'] for item in nobel_data)
print(f"Unique categories: {unique_titles}")
```

Listing 3: Extracting data into a structured format

7.5.4 Step 4: Convert to pandas DataFrame

```
import pandas as pd

# Convert list of dictionaries to DataFrame
df = pd.DataFrame(nobel_data)

# View the first few rows
```

```
print(df.head())

# Save to CSV file
df.to_csv('nobel_prizes.csv', index=False)
```

Listing 4: Creating a DataFrame

8 Frequently Asked Questions

- **Q: Can I audit this course?**
- A: Yes, but if you audit, you cannot take the course for credit later.
- **Q: Can I take the course asynchronously (watching recordings only)?**
- A: **College students**: No. **Graduate students**: Not recommended. With less than 33% attendance, your maximum grade is B.
- **Q: I'm missing some prerequisites. Should I still take this course?**
- A: Complete HW0 and see how you do. Math/stats gaps can be addressed with TF help. However, if you have **zero programming experience**, strongly consider postponing to next year.
- **Q: Can I reschedule the midterm for travel?**
- A: No. The midterm (week of Oct 22-24) and final exam (Dec 11) dates are fixed. Plan accordingly.
- **Q: Can I use my own project idea?**
- A: Yes, but the data must be **public** (shareable with your group), and you must work in a group of 3-5 students.
- **Q: What if I miss a class?**
- A: All lectures are recorded and available on Canvas. Missing one class won't affect your grade. Just maintain at least 66% attendance to be eligible for an A.

9 Key Takeaways

Key Summary

Summary of Lecture 01

What is Data Science?

- An interdisciplinary field combining math/statistics, computer science, and domain knowledge
- Represents a paradigm shift: using data + computing to understand the world, sometimes bypassing traditional equations
- Has tremendous potential (medical diagnosis, drug discovery) but also serious risks (bias, ethics)

The Data Science Process:

1. Ask an interesting question (hypothesis-driven)
2. Get the data (ethically and legally)
3. Explore the data (EDA—don't skip this!)
4. Model the data (build, fit, validate)
5. Communicate results (tell the story)

Course Philosophy:

- "Wax on, wax off"—master the fundamentals before building complex systems
- Don't just use `.fit()`—understand what's happening inside
- Balance ML optimization with statistical understanding

Critical Logistics:

- Attendance is required: 66% minimum for A eligibility
- Prerequisites: Programming experience is essential; math/stats gaps can be filled
- Pair work encouraged for homeworks

CS109A: Introduction to Data Science
Lecture 02: Data and Visualization

Harvard University

Fall 2024

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 02: Data and Visualization
- **Instructor:** Kevin Rader
- **Objective:** Understand data types, collection methods, descriptive statistics, and visualization techniques for exploratory data analysis

Key Summary

This lecture covers the fundamental building blocks of data science: understanding what data are, where they come from, how to structure them for analysis, and how to explore them effectively. We cover the distinction between populations and samples, measures of center (mean, median, mode) and spread (variance, standard deviation), and the critical importance of data visualization. The famous Anscombe's Quartet demonstrates why you should **always visualize your data** rather than relying solely on summary statistics. We also explore various visualization techniques including histograms, bar plots, scatter plots, box plots, and violin plots.

Contents

1 Key Terminology

Before diving into data analysis, you need to understand these fundamental terms:

Table 1: *Essential Data Science Terminology*

Term	Simple Explanation	Notes
Data	A collection of facts, values, or information obtained through observation	Plural form; singular is "datum"
Tabular Data	Data organized in rows and columns like a spreadsheet	Also called "Tidy Data"
Observation	A single unit of analysis (one row)	Example: one movie, one student
Variable	A characteristic being measured (one column)	Also called "feature" or "attribute"
Population	The entire group you want to study	Example: ALL students in this class
Sample	A subset taken from the population	Example: Students who attended today
EDA	Exploratory Data Analysis	Using visuals and statistics to find patterns
Measures of Center		
Mean ($\bar{x}$)	Sum of all values divided by count	Sensitive to outliers
Median	The middle value when data is sorted	Robust to outliers
Mode	Most frequently occurring value	Used for categorical data
Measures of Spread		
Variance (s^2)	Average squared distance from the mean	Units are squared
Std. Dev. (s)	Square root of variance	Same units as original data

2 What is Data?

Data science starts with **data**. But what exactly is data?

2.1 Definition of Data

Definition:

Data vs. Datum

- **Datum** (singular): A single piece of information or measurement
- **Data** (plural): A collection of information pieces obtained through observation or measurement

Data can be:

- **Numeric**: Numbers (integers, decimals)
- **Categorical**: Groups or categories ("male/female", "red/blue/green")
- **Boolean**: True/False, Yes/No, 1/0
- **Strings**: Text ("Hello World")

In the modern world, **everything is data**. Facebook collects your social interactions, Google tracks your searches, your phone records your location, and even your grocery store tracks your purchases.

2.2 Where Does Data Come From?

Data can come from three main sources:

2.2.1 1. Internal Sources (Primary Data)

Data you or your organization collect directly:

- Scientific experiments
- Clinical trials
- Surveys you design and distribute
- Company sales records

2.2.2 2. Existing External Sources

Data that someone else has already collected and made available:

- Government open data portals
- Kaggle datasets
- Sports statistics websites
- Academic research databases

2.2.3 3. External Sources Requiring Collection

Data that exists online but requires effort to extract:

- **APIs**: Official interfaces provided by companies
- **RSS Feeds**: Streams from blogs and news sites
- **Web Scraping**: Extracting data from HTML pages

2.3 Methods of Online Data Collection

Table 2: *Three Methods of Online Data Collection*

Method	Description	Pros/Cons
API	Official interface provided by companies (Google Maps, Twitter, Spotify)	Pro: Legal, stable, accurate data. Con: Often rate-limited or paid.
RSS	Streams of updated content from blogs/news sites	Pro: Free, real-time updates. Con: Limited to what publishers provide.
Web Scraping	Extracting data directly from HTML code	Pro: Flexible, free. Con: Legal/ethical concerns, fragile.

Warning

Ethical and Legal Concerns with Web Scraping

Web scraping is powerful but comes with serious responsibilities:

- **Terms of Service**: Many websites explicitly prohibit scraping
- **Privacy**: Never collect or publish private/personal information
- **Server Load**: Excessive scraping can overwhelm servers (similar to a DoS attack)
- **Potential Harm**: Ask yourself: "Could publishing this data harm someone?"

Rule of thumb: Just because data is publicly available doesn't mean you can use it however you want. Always ask "Should I be doing this?" before scraping.

3 Data Types and Structures

3.1 Atomic Data Types

The most basic building blocks of data:

- **Numeric**:
  - **Integers**: Whole numbers (e.g., 42, -7, 0)
  - **Floats**: Decimal numbers (e.g., 3.14, -0.001)
- **Boolean**: True/False values (1/0, Yes/No)
- **Strings**: Text sequences (e.g., "Hello", "CS109A")

3.2 Compound Data Types

More complex structures built from atomic types:

- **Lists**: Ordered sequences of values

```
my_list = [1, 2, 3, 4, 5]
```

- **Dictionaries**: Key-value pairs

```
student = {
    "first": "Kevin",
    "last": "Rader",
    "classes": ["CS109A", "STAT104"]
}
```

3.3 Tabular Data: The Gold Standard

Key Information

Why Tabular Data Matters

Most data analysis tools (pandas, sklearn, etc.) expect your data in **tabular format**—like an Excel spreadsheet with rows and columns.

- **Rows = Observations**: Each row represents one unit of analysis (one movie, one student, one transaction)
- **Columns = Variables**: Each column represents one characteristic being measured (rating, age, price)

```
import pandas as pd

# Read a CSV file into a DataFrame
imdb = pd.read_csv('imdb_top_1000.csv')

# View the first 5 rows
imdb.head()
```

Listing 1: Loading tabular data with pandas

The output shows a table where:

- Each **row** is a different movie (The Shawshank Redemption, The Godfather, etc.)
- Each **column** is a variable (Series_Title, Released_Year, IMDB_Rating, etc.)

3.4 Variable Types: A Critical Distinction

Important:

Why Variable Types Matter The type of variable determines:

- Which **summary statistics** you can calculate
- Which **visualizations** are appropriate
- Which **models** you can use

You can calculate the mean of "height" but not the mean of "favorite color"!

Variables are divided into two main categories:

3.4.1 Quantitative (Numeric) Variables

Values are numbers where arithmetic operations make sense.

- **Discrete**: Values are countable, typically integers
  - Examples: Number of siblings, dice rolls, page views
- **Continuous**: Values can be any number within a range
  - Examples: Height, weight, temperature, time

3.4.2 Categorical Variables

Values fall into distinct groups or categories.

- **Nominal**: No natural ordering
  - Examples: Blood type (A, B, O, AB), pet preference (dog, cat, rat)
- **Ordinal**: Natural ordering exists
  - Examples: Letter grades (A, B, C, D, F), satisfaction (high, medium, low)

3.5 Messy Data and Tidy Data

Real-world data is rarely clean. Common problems include:

- **Missing values**: Empty cells
- **Wrong values**: Data entry errors (age = 200)
- **Format mismatches**: Different date formats, inconsistent naming
- **Messy structure**: Data not in tabular format

Example:

Converting Messy Data to Tidy Data **BEFORE (Messy):** Weekend produce deliveries

	Friday	**Saturday**	**Sunday**
Morning	15	158	10
Afternoon	2	90	20
Evening	55	12	45

Problems:

- One row contains 3 observations (Friday morning, Saturday morning, Sunday morning)
- "Friday" is a value, not a variable name
- Hard to calculate "average deliveries" or "total by day"

AFTER (Tidy):

ID	**Time**	**Day**	**Deliveries**
1	Morning	Friday	15
2	Morning	Saturday	158
3	Morning	Sunday	10
4	Afternoon	Friday	2
5	Afternoon	Saturday	90
...	...	...	...
9	Evening	Sunday	45

Why this is better:

- **1 observation = 1 row**: Each row is a unique time-day combination
- **1 variable = 1 column**: Time, Day, and Deliveries are separate columns
- Easy to filter, group, and analyze with pandas

4 Population vs. Sample

Understanding the distinction between population and sample is fundamental to all of statistics.

Definition:

Population and Sample

- **Population**: The *entire* set of objects/individuals you want to study
  - Example: ALL students enrolled in CS109A
- **Sample**: A *subset* of the population that you actually observe
  - Example: Students who attended class today

We analyze the **sample** to make inferences about the **population**.

4.1 Sampling Bias

The sample must be **representative** of the population. When it isn't, we have **sampling bias**.

Table 3: *Types of Sampling Bias*

Type	Description
Selection Bias	Some individuals are more likely to be selected than others
Non-response Bias	People who don't respond may be systematically different from those who do
Volunteer Bias	People who volunteer may be more enthusiastic or have stronger opinions

Example:

Sampling Bias in Practice **Example 1: Class Attendance**

- **Goal**: Survey satisfaction of ALL CS109A students
- **Sample**: Only students who came to class today
- **Problem**: Students who skip class might be less satisfied—their opinions are missing!
- **Result**: Overestimated satisfaction scores

Example 2: Early Adopters

- **Goal**: Test if a new app feature works for all users
- **Sample**: "Early adopters" who signed up for beta testing
- **Problem**: Early adopters are tech-savvy and excited about new features
- **Result**: Feature seems great in testing, but flops when released to everyone

Lesson: Always ask "Who is missing from my sample?" and "How might they be different?"

5 Descriptive Statistics: Measures of Center

Descriptive statistics summarize the characteristics of your data. We start with measures of **center**—where the "typical" value is located.

5.1 Mean (Average)

Definition:

Sample Mean The **mean** is the sum of all values divided by the count:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

Intuition: The mean is the "balancing point" of the distribution—if you placed the data on a seesaw, the mean is where it would balance.

Warning

The Mean is Sensitive to Outliers

Consider these two datasets:

- Dataset A: [1, 2, 3, 4, 5] → Mean = 3
- Dataset B: [1, 2, 3, 4, **100**] → Mean = 22

One extreme value (100) pulled the mean from 3 to 22! This is why the mean can be misleading when outliers are present.

5.2 Median

Definition:

Sample Median The **median** is the middle value when data is sorted:

- If n is odd: The middle value
- If n is even: Average of the two middle values

Advantage: The median is **robust** to outliers—extreme values don't affect it much.

Example:

Mean vs. Median Student ages: [17, 19, 21, 22, 23, 24, 26, **38**]

Median: Average of 22 and 23 = **22.5**

Mean: $(17 + 19 + 21 + 22 + 23 + 24 + 26 + 38)/8 = \mathbf{23.75}$

The 38-year-old graduate student pulls the mean up, but the median barely moves!

5.3 Mean vs. Median: Skewness

The relationship between mean and median tells you about the **shape** of the distribution:

Table 4: *Mean, Median, and Distribution Shape*

Distribution Shape	Relationship	Example
Symmetric	Mean $\approx$ Median	Normal (bell curve)
Right-skewed	Mean $>$ Median	Income distribution
Left-skewed	Mean $<$ Median	Age at retirement

Key Information

Why Financial Data Uses Medians

Income and housing prices are typically **right-skewed**—most people earn moderate amounts, but a few billionaires earn enormously more.

If we reported mean income, those billionaires would make everyone seem richer than they are. That's why economists report **median household income**—it better represents the "typical" household.

5.4 Mode

Definition:

Mode The **mode** is the most frequently occurring value in a dataset.

Use case: Primarily for **categorical data** where mean and median don't make sense.

Example:

Mode for Categorical Data Favorite pets: ["Dog", "Cat", "Dog", "Rat", "Cat", "Dog"]

Mode: "Dog" (appears 3 times)

You can't calculate the "mean" of pet preferences, but you can find the most popular choice!

5.5 Computational Efficiency

Key Information

Mean is Faster to Compute Than Median

- **Mean**: $O(n)$ — Just scan through once, keeping track of sum and count
- **Median**: $O(n \log n)$ — Requires sorting first!

When you have billions of data points (like Facebook or Google), this difference matters. This is one reason why means are more commonly reported even when medians might be more appropriate.

6 Descriptive Statistics: Measures of Spread

Knowing the center isn't enough—we also need to know how **spread out** the data is.

6.1 Range

Definition:

Range

$$\text{Range} = \text{Maximum} - \text{Minimum}$$

Problem: Only uses two values (max and min), ignoring everything in between.

6.2 Variance

Definition:

Sample Variance The **variance** measures the average squared distance from the mean:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Steps:

1. Calculate the mean ($\bar{x}$)
2. For each value, find the distance from the mean ($x_i - \bar{x}$)
3. Square each distance (removes negatives, emphasizes larger deviations)
4. Take the average of squared distances (dividing by $n-1$)

Example:

Why $n-1$ Instead of n? **Question**: Why do we divide by $n-1$ instead of n?
Simple Intuition: What's the minimum number of observations needed to measure "spread"? With just **one observation** (e.g., [5]), you can't say anything about how spread out the data is! The formula with $n-1$ agrees: $\frac{1}{1-1} = \frac{1}{0}$ is undefined.
You need at least 2 observations to talk about spread.
Technical Answer: Dividing by $n-1$ (called "degrees of freedom") corrects for a bias that occurs because we estimate the mean from the same data. This gives us an "unbiased estimator" of the true population variance.

6.3 Standard Deviation

Definition:

Sample Standard Deviation

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Key advantage: The standard deviation has the **same units as the original data**, making it interpretable.

If heights are measured in cm, variance is in cm^2 (meaningless), but standard deviation is back in cm!

Key Information

Interpreting Standard Deviation

The standard deviation is roughly the "average distance" from the mean.

If the mean height is 170 cm and the standard deviation is 10 cm, most people are within about 10 cm of 170 cm (between 160–180 cm).

7 Why Visualization Matters: Anscombe's Quartet

Important:

Never Trust Summary Statistics Alone! **Anscombe's Quartet** is a famous example showing why you must **always visualize your data**.

Four different datasets have:

- Same mean of X (9.0)
- Same mean of Y (7.50)
- Same variance of X (11.0)
- Same variance of Y (4.12)
- Same correlation (0.816)

Based on these statistics, the datasets seem identical. But when you plot them...

The Four Datasets (with identical summary statistics):

Dataset I		Dataset II		Dataset III		Dataset IV	
X	Y	X	Y	X	Y	X	Y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

What the scatter plots reveal:

- **Dataset I**: Linear relationship with random scatter (what you'd expect)
- **Dataset II**: Perfect **parabola**—clearly nonlinear!
- **Dataset III**: Perfect line except for one **Y-outlier**
- **Dataset IV**: All X values are 8, except one **X-outlier** at 19

Key Summary

The Lesson of Anscombe's Quartet

Always visualize your data before modeling. Summary statistics can hide crucial patterns:

- Nonlinear relationships
- Outliers
- Clusters or subgroups
- Data quality issues

Don't just calculate R^2 and call it a day!

8 Basic Visualization Types

Different visualizations serve different purposes. Choose based on what you want to learn.

8.1 Visualizations by Purpose

Table 5: *Choosing the Right Visualization*

Chart Type	Purpose	Variables	What to Look For
Histogram	Distribution of one numeric variable	1 numeric	Shape, center, spread, outliers
Bar Plot	Frequency of categories	1 categorical	Which categories are most/least common
Scatter Plot	Relationship between two variables	2 numeric	Direction, strength, form, outliers
Box Plot	Compare distributions across groups	1 numeric + 1 categorical	Median, IQR, outliers by group
Violin Plot	Compare distribution shapes across groups	1 numeric + 1 categorical	Full distribution shape (bimodality, etc.)
KDE Plot	Smooth distribution curve	1 numeric	Avoids bin-width problems of histograms

8.2 Histogram vs. Bar Plot

Both use bars, but they're fundamentally different:

Table 6: *Histogram vs. Bar Plot*

Feature	Histogram	Bar Plot
Variable type	Numeric (continuous)	Categorical
Bars	Touch each other	Separated
X-axis	Has natural order	Order is arbitrary
Purpose	Show distribution shape	Compare category frequencies

8.3 Histogram Bin Width Matters

Warning

The appearance of a histogram depends heavily on **bin width**:

- **Too wide**: Loses detail, everything looks uniform
- **Too narrow**: Too much noise, hard to see patterns
- **Just right**: Shows the true shape of the distribution

Always try multiple bin widths to make sure you're not missing important patterns!

Alternative: Use a **Kernel Density Estimate (KDE)** which creates a smooth curve and avoids the bin-width problem.

8.4 Scatter Plots: Reading Relationships

When looking at a scatter plot, ask four questions:

1. **Direction**: Is the relationship positive (both increase together) or negative (one increases as the other decreases)?
2. **Strength**: How tightly do points cluster around the trend? (Strong = tight, Weak = scattered)
3. **Form**: Is the relationship linear or curved?
4. **Outliers**: Are there any points that don't fit the pattern?

8.5 Box Plots: Comparing Distributions

A box plot shows five key statistics:

- **Median**: Line in the middle of the box
- **Q1 and Q3**: Bottom and top of the box (middle 50% of data)
- **Whiskers**: Extend to the smallest/largest non-outlier values
- **Outliers**: Individual points beyond the whiskers

Use case: Comparing how a numeric variable differs across categories (e.g., income by education level).

8.6 Violin Plots: More Detail Than Box Plots

A violin plot is a box plot combined with a KDE:

- Shows the full distribution shape on both sides
- Reveals bimodality or unusual shapes that box plots hide
- More informative but takes more space

8.7 Why Pie Charts Are Usually Bad

Warning

The Problem with Pie Charts
Humans are bad at comparing angles. When pie slices are similar sizes, it's nearly impossible to tell which is bigger.
Better alternative: Use a bar plot. Humans easily compare bar heights.
Exception: Pie charts work when you want to emphasize "parts of a whole" and there are only 2–3 categories with very different sizes.

9 Visualizing Multiple Variables

What if you have more than two variables? Three-dimensional scatter plots are tempting but usually fail on 2D screens.

9.1 The Problem with 3D Scatter Plots

A "scatter cloud" is nearly impossible to interpret:

- Depth perception is lost on a flat screen
- Occlusion: points hide behind other points
- You can't rotate the view interactively in a static report

9.2 Better Approaches: Aesthetic Mappings

Instead of adding a third spatial dimension, map additional variables to visual properties:

- **Color**: Great for categorical variables (up to 5–8 groups)
- **Size**: Good for numeric variables
- **Shape**: Good for categorical variables (2–3 groups max)
- **Animation**: Good for time (each frame = one time point)

Example:

Gapminder: 5 Variables in One Plot The famous "Wealth and Health of Nations" visualization shows:

1. **X-axis**: Income per person
2. **Y-axis**: Life expectancy
3. **Circle size**: Population
4. **Circle color**: Continent
5. **Animation**: Year (time)

Key insights visible:

- Positive (but nonlinear) relationship between income and life expectancy
- China's dramatic rise from poverty to prosperity
- The US has lower life expectancy than peers with similar income

Design choices:

- Population → Size (numeric → size works well)
- Continent → Color (categorical → distinct colors)

10 Historical Examples of Data Visualization

Data visualization isn't new—some of history's most important discoveries came from visualizing data.

10.1 John Snow's Cholera Map (1854)

- **Problem**: Cholera outbreak in London—source unknown
- **Visualization**: Dot map showing location of each cholera death
- **Discovery**: Deaths clustered around the Broad Street water pump
- **Action**: Removed the pump handle; outbreak ended
- **Legacy**: Proved that cholera spread through contaminated water, not "bad air"

10.2 Florence Nightingale's Rose Chart (1858)

- **Problem**: High death rates in military hospitals during the Crimean War
- **Visualization**: Rose chart showing deaths by cause over time
- **Discovery**: Blue (preventable disease) vastly exceeded red (combat wounds)
- **Action**: Hospital sanitation reforms
- **Legacy**: Pioneered the use of statistics in healthcare policy

10.3 Charles Minard's Napoleon Map (1869)

- **Subject**: Napoleon's disastrous invasion of Russia (1812)
- **Variables shown**: Army size (line width), geography (map), direction (color), time, and temperature
- **Story**: 422,000 troops entered; only 10,000 returned
- **Legacy**: Called "the best statistical graphic ever drawn"

11 Principles of Effective Visualization

11.1 1. Graphical Integrity

Don't lie with your visuals.

- Y-axis should usually start at zero for bar charts
- Geographic maps should account for population, not just area
- Don't use 3D effects that distort perception

11.2 2. Keep It Simple

Avoid "chart junk"—unnecessary decoration that doesn't help understanding.

- No 3D effects on 2D data
- No excessive gridlines or backgrounds
- Maximize the "data-ink ratio" (information per pixel)

11.3 3. Use the Right Display

Visual channels have different effectiveness for encoding data:

1. Position (most accurate)
2. Length
3. Angle (pie charts are here—not great!)
4. Area
5. Color intensity
6. Shape (least accurate for quantitative data)

11.4 4. Use Color Strategically

- **Qualitative**: Distinct colors for categories (limit to 5–8)
- **Sequential**: Light-to-dark for ordered values
- **Diverging**: Two colors from a neutral center (e.g., blue–white–red)
- **Avoid**: Rainbow color maps (no natural order)
- **Consider**: Color blindness (avoid red-green combinations)

11.5 5. Know Your Audience

- **Exploratory**: For yourself—neutral, finding patterns
- **Explanatory**: For others—making a specific point

12 Key Takeaways

Key Summary

Summary of Lecture 02

What is Data?

- Data = collected information; tabular format is ideal
- Variables can be numeric (discrete/continuous) or categorical (nominal/ordinal)
- Always consider: Where did this data come from? What might be missing?

Population vs. Sample

- Population = everyone you care about
- Sample = the subset you actually observe
- Watch out for sampling bias (selection, non-response, volunteer)

Measures of Center

- Mean: Balancing point (sensitive to outliers)
- Median: Middle value (robust to outliers)
- Mode: Most frequent (for categorical data)
- If Mean > Median: Right-skewed distribution

Measures of Spread

- Variance: Average squared deviation (units are squared)
- Standard Deviation: Square root of variance (same units as data)

Visualization

- Anscombe's Quartet: ALWAYS visualize before modeling!
- Choose the right chart for your purpose
- Histograms/KDE for distributions, scatter plots for relationships, box/violin plots for comparisons
- Multiple variables: Use color, size, shape, animation
- Follow principles: integrity, simplicity, appropriate display, strategic color

CS109A: Introduction to Data Science
Lecture 03: Introduction to Pandas

Harvard University

Fall 2024

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 03: Introduction to Pandas
- **Instructor:** Chris Gumb
- **Objective:** Learn the fundamentals of Pandas for data manipulation and analysis in Python

Key Summary

This lecture introduces **Pandas**, the essential Python library for data manipulation and analysis. You'll learn about the two core data structures: `Series` (1-dimensional) and `DataFrame` (2-dimensional tables). We cover how to create these structures, load data from CSV files, inspect and clean data, select subsets using boolean indexing and the `loc`/`iloc` accessors, and perform aggregations with `groupby`. By the end of this lecture, you'll have the foundational skills needed to work with tabular data in Python.

Contents

1 Introduction: Why Pandas?

1.1 What is Pandas?

Definition:

Pandas **Pandas** is a Python library that provides high-performance, easy-to-use data structures and data analysis tools. The name comes from "Panel Data" (a term from econometrics) and "Python Data Analysis."

Pandas gives you access to data types that don't exist in standard Python:

- **Series**: A 1-dimensional labeled array
- **DataFrame**: A 2-dimensional labeled table (like an Excel spreadsheet)

1.2 Why Use Pandas Instead of Plain Python?

You could store data in Python lists and dictionaries, but Pandas offers significant advantages:

- **Speed**: Pandas is built on NumPy, which stores data in contiguous memory blocks for fast operations
- **Convenience**: Methods like `.head()`, `.describe()`, `.groupby()` make common operations trivial
- **Data Alignment**: Pandas automatically aligns data by labels during operations
- **Missing Data Handling**: Built-in support for missing values (`NaN`)
- **File I/O**: Easy reading/writing of CSV, Excel, SQL databases, JSON, and more

1.3 Importing Pandas

The standard convention is to import pandas with the alias `pd`:

```
1 import pandas as pd
2 import numpy as np  # Often used alongside pandas
```

Warning

Don't break conventions! Always use `pd` for pandas and `np` for numpy. Using other aliases will confuse anyone reading your code.

2 The Series Data Structure

2.1 What is a Series?

Definition:

Series A **Series** is a one-dimensional array with labels (called an **index**). Think of it as:

- A single column from a spreadsheet
- A dictionary where keys are the index and values are the data
- A NumPy array with attached labels

2.2 Creating a Series

The simplest way is from a Python list:

```
# Create a simple Series
my_list = [10, 20, 30, 40, 50]
s = pd.Series(my_list)
print(s)
```

Listing 1: Creating a Series from a list

Output:

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

Notice the output shows:

- **Left column**: The **index** (0, 1, 2, 3, 4)—created automatically
- **Right column**: The **values** from your list
- **Bottom**: The **dtype** (data type)—`int64` means 64-bit integers

2.3 Series Attributes

Every Series has four key attributes:

```
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'], name='my_series')

print(s.values)    # The actual data (as a NumPy array)
# Output: array([10, 20, 30])

print(s.index)     # The labels
# Output: Index(['a', 'b', 'c'], dtype='object')
```

```
8
9  print(s.name)      # The series name (becomes column name in DataFrame)
10 # Output: 'my_series'
11
12 print(s.dtype)     # The data type
13 # Output: dtype('int64')
```

Listing 2: Series attributes

2.4 Custom Indexes

The index doesn't have to be integers—you can use any labels:

```
s = pd.Series([100, 200, 300])
s.index = ['apple', 'banana', 'cherry']
print(s)

# Output:
# apple      100
# banana     200
# cherry     300
# dtype: int64

# Access by label
print(s['banana'])  # Output: 200
```

Listing 3: Custom string index

Warning

Index Labels Can Be Non-Unique!

Unlike dictionary keys, Series indices can have duplicates:

```
s.index = ['a', 'a', 'b']
print(s['a'])
# Returns BOTH values with index 'a':
# a    100
# a    200
```

This can cause unexpected behavior if you're not careful!

3 The DataFrame Data Structure

3.1 What is a DataFrame?

Definition:

DataFrame A **DataFrame** is a 2-dimensional labeled data structure—like an Excel spreadsheet or SQL table.

- Each **column** is a Series
- All columns share the same **index** (row labels)
- Different columns can have different data types

3.2 Creating a DataFrame

3.2.1 Method 1: From a List of Dictionaries (Row by Row)

Each dictionary represents one row; keys become column names:

```
data = [
    {'fruit': 'apple', 'color': 'red', 'price': 1.50},
    {'fruit': 'banana', 'color': 'yellow', 'price': 0.75},
    {'fruit': 'cherry', 'color': 'red', 'price': 3.00}
]

df = pd.DataFrame(data)
print(df)
```

Listing 4: DataFrame from list of dictionaries

Output:

```
    fruit   color  price
0   apple     red   1.50
1  banana  yellow   0.75
2  cherry     red   3.00
```

3.2.2 Method 2: From a Dictionary of Lists (Column by Column)

Each key is a column name; the list becomes that column's values:

```
data = {
    'fruit': ['apple', 'banana', 'cherry'],
    'color': ['red', 'yellow', 'red'],
    'price': [1.50, 0.75, 3.00]
}

df = pd.DataFrame(data)
# Same result as above
```

Listing 5: DataFrame from dictionary of lists

Key Information

Which Method to Use?

- **List of dictionaries**: Useful when scraping data (you build up rows one at a time)
- **Dictionary of lists**: Useful when you already have column-wise data

3.3 DataFrame Attributes

```python
print(df.shape)     # (rows, columns)
# Output: (3, 3)

print(df.columns)  # Column names
# Output: Index(['fruit', 'color', 'price'], dtype='object')

print(df.index)     # Row labels
# Output: RangeIndex(start=0, stop=3, step=1)

print(df.dtypes)    # Data type of each column
# Output:
# fruit      object
# color      object
# price     float64
# dtype: object
```

Listing 6: DataFrame attributes

4 Loading Data from Files

4.1 Reading CSV Files

The most common way to get data into pandas:

```
# Read a CSV file into a DataFrame
df = pd.read_csv('data/survey_results.csv')

# Peek at the first few rows
df.head()
```

Listing 7: Reading a CSV file

Key Information

Other File Formats

Pandas can read many formats:

- `pd.read_excel('file.xlsx')` — Excel files
- `pd.read_json('file.json')` — JSON files
- `pd.read_html('url')` — Tables from web pages
- `pd.read_sql(query, connection)` — SQL databases
- `pd.read_clipboard()` — From your clipboard!

4.2 Inspecting Your Data

After loading data, always inspect it first:

```
# First 5 rows (default) or specify n
df.head()
df.head(2)

# Last 5 rows
df.tail()

# Shape: (rows, columns)
df.shape

# Column names
df.columns

# Data types and memory usage
df.info()

# Summary statistics for numeric columns
df.describe()

# Data types only
df.dtypes
```

Listing 8: Inspecting a DataFrame

5 Selecting Data

5.1 Selecting Columns

```python
# Single column (returns a Series)
df['fruit']

# Single column using dot notation (shortcut)
df.fruit  # Only works if column name has no spaces

# Multiple columns (returns a DataFrame)
df[['fruit', 'price']]
```

Listing 9: Selecting columns

Warning

Be Careful with Dot Notation

`df.fruit` is convenient but dangerous if:

- Column name has spaces (`df.my column` doesn't work)
- Column name matches a DataFrame method (`df.head` returns the method, not a column!)

The bracket notation `df['fruit']` is always safe.

5.2 The loc and iloc Accessors

Pandas provides two ways to select rows:

Definition:

loc vs iloc

- **loc**: Selection by **label** (what you see in the index)
- **iloc**: Selection by **integer position** (0, 1, 2, ...)

```python
# Create a DataFrame with a non-default index
df = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Charlie'],
    'age': [25, 30, 35]
}, index=['a', 'b', 'c'])

# loc: by label
df.loc['b']         # Row with label 'b'
df.loc['a':'b']     # Rows 'a' through 'b' (inclusive!)
df.loc['a', 'age'] # Specific cell: row 'a', column 'age'

# iloc: by position
df.iloc[1]          # Second row (position 1)
df.iloc[0:2]        # First two rows (exclusive end!)
df.iloc[0, 1]       # Specific cell: row 0, column 1
```

Listing 10: Using loc and iloc

Important:

The Index Trap After filtering or sorting, row indices get scrambled:

```
df_sorted = df.sort_values('age', ascending=False)
print(df_sorted)
#       name  age
# c   Charlie   35
# b       Bob   30
# a     Alice   25

# Trying to get "first row" by position
df_sorted.iloc[0]  # Gets Charlie (correct)

# Trying by label 0
df_sorted.loc[0]   # ERROR! No label '0' exists
```

Solution: Use `.reset_index(drop=True)` after sorting/filtering.

6 Boolean Indexing (Filtering)

6.1 Creating Boolean Masks

A **boolean mask** is a Series of True/False values that you use to filter rows:

```
df = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Charlie', 'Diana'],
    'age': [25, 30, 35, 28],
    'city': ['NYC', 'LA', 'NYC', 'LA']
})

# Create a mask: which rows have age > 27?
mask = df['age'] > 27
print(mask)
# 0    False
# 1     True
# 2     True
# 3     True
# dtype: bool
```

Listing 11: Creating boolean masks

6.2 Applying the Mask

Use the mask inside brackets to filter:

```
# Get rows where age > 27
df[mask]
# or directly:
df[df['age'] > 27]

#       name  age city
# 1      Bob   30   LA
# 2  Charlie   35  NYC
# 3    Diana   28   LA
```

Listing 12: Filtering with boolean masks

6.3 Combining Conditions

```
# Use & for AND, | for OR, ~ for NOT
# IMPORTANT: Each condition must be in parentheses!

# Age > 27 AND city is NYC
df[(df['age'] > 27) & (df['city'] == 'NYC')]

# Age > 30 OR city is LA
df[(df['age'] > 30) | (df['city'] == 'LA')]

```

```
# NOT in NYC
df[~(df['city'] == 'NYC')]
```

Listing 13: Combining conditions

Warning

Parentheses are Required!

Due to Python operator precedence, you must wrap each condition in parentheses:

```
# WRONG:
df[df['age'] > 27 & df['city'] == 'NYC']  # Error!

# CORRECT:
df[(df['age'] > 27) & (df['city'] == 'NYC')]
```

7 Common DataFrame Operations

7.1 Sorting

```python
# Sort by one column
df.sort_values('age')

# Sort descending
df.sort_values('age', ascending=False)

# Sort by multiple columns
df.sort_values(['city', 'age'])

# Reset index after sorting (usually what you want)
df.sort_values('age').reset_index(drop=True)
```

Listing 14: Sorting DataFrames

7.2 Value Counts

Count occurrences of unique values:

```python
# How many people in each city?
df['city'].value_counts()
# NYC    2
# LA     2

# Get unique values
df['city'].unique()
# array(['NYC', 'LA'], dtype=object)

# Number of unique values
df['city'].nunique()
# 2
```

Listing 15: Counting values

7.3 Group By and Aggregation

Definition:

GroupBy **GroupBy** splits data into groups based on a column's values, then applies an aggregation function to each group.

```python
# Average age by city
df.groupby('city')['age'].mean()
# city
# LA     29.0
# NYC    30.0
```

```
6
7  # Multiple aggregations
8  df.groupby('city').agg({
9      'age': ['mean', 'min', 'max'],
10     'name': 'count'
11 })
12
13 # Count rows per group
14 df.groupby('city').size()
```

Listing 16: GroupBy operations

7.4 Handling Missing Data

```python
# Check for missing values
df.isna()            # Boolean mask of NaN locations
df.isna().sum()      # Count NaNs per column

# Drop rows with any NaN
df.dropna()

# Fill NaN with a value
df.fillna(0)
df['column'].fillna('Unknown')

# Fill with column mean
df['age'].fillna(df['age'].mean())
```

Listing 17: Working with missing values

7.5 Renaming Columns

```python
# Rename specific columns
df.rename(columns={'old_name': 'new_name'})

# Rename all columns at once
df.columns = ['col1', 'col2', 'col3', 'col4']

# Make all column names lowercase
df.columns = df.columns.str.lower()
```

Listing 18: Renaming columns

8 Method Chaining

8.1 What is Method Chaining?

Instead of saving intermediate results to variables, you can chain multiple operations together:

```
# Without chaining (verbose)
df_filtered = df[df['age'] > 25]
df_sorted = df_filtered.sort_values('age')
df_top = df_sorted.head(3)

# With chaining (concise)
result = (df[df['age'] > 25]
          .sort_values('age')
          .head(3))
```

Listing 19: Method chaining example

Key Information

Benefits of Method Chaining

- More concise code
- No intermediate variables cluttering your namespace
- Reads like a pipeline: "Take df, filter it, sort it, take top 3"

Tip: Wrapping in parentheses lets you break across multiple lines for readability.

9 Practical Example: Survey Data

Let's walk through a realistic example using class survey data:

```python
# 1. Load the data
df = pd.read_csv('data/survey_raw.csv')

# 2. Inspect
print(df.shape)
print(df.columns)
df.head()

# 3. Clean column names (remove spaces, lowercase)
df.columns = ['timestamp', 'program', 'jupyter_exp',
              'python_exp', 'pandas_skill', 'os',
              'dark_mode', 'languages', 'dob']

# 4. Check data types
df.dtypes
df.info()

# 5. Convert dark_mode to boolean
df['dark_mode'] = df['dark_mode'] == 'Yes'

# 6. Filter to students with pandas skill > 3
skilled = df[df['pandas_skill'] > 3]

# 7. Count programs
df['program'].value_counts()

# 8. Average pandas skill by program
(df.groupby('program')['pandas_skill']
   .mean()
   .sort_values(ascending=False))

# 9. Save cleaned data
df.to_csv('data/survey_clean.csv', index=False)
```

Listing 20: Complete example workflow

10 Advanced: Exploding Lists

Sometimes a column contains comma-separated values that you want to split:

```python
# Sample data
df = pd.DataFrame({
    'name': ['Alice', 'Bob'],
    'languages': ['English, Spanish', 'English, Mandarin, French']
})

# Step 1: Split the string into a list
df['lang_list'] = df['languages'].str.split(', ')

# Step 2: Explode - each list item becomes its own row
df_exploded = df.explode('lang_list')

#     name                    languages    lang_list
# 0  Alice           English, Spanish      English
# 0  Alice           English, Spanish      Spanish
# 1    Bob  English, Mandarin, French      English
# 1    Bob  English, Mandarin, French     Mandarin
# 1    Bob  English, Mandarin, French       French

# Now you can count languages, filter, etc.
df_exploded['lang_list'].value_counts()
```

Listing 21: Exploding comma-separated values

11 Key Takeaways

Key Summary

Summary of Lecture 03: Pandas Fundamentals

Core Data Structures

- **Series**: 1D labeled array (values + index + name + dtype)
- **DataFrame**: 2D table of Series (columns share an index)

Creating DataFrames

- `pd.DataFrame(list_of_dicts)` — Row by row
- `pd.DataFrame(dict_of_lists)` — Column by column
- `pd.read_csv('file.csv')` — From file

Inspecting Data

- `df.head()`, `df.tail()`, `df.shape`
- `df.info()`, `df.describe()`, `df.dtypes`

Selecting Data

- Columns: `df['col']` or `df[['col1', 'col2']]`
- Rows by label: `df.loc['label']`
- Rows by position: `df.iloc[0]`
- Boolean filtering: `df[df['col'] > value]`

Common Operations

- Sort: `df.sort_values('col')`
- Count: `df['col'].value_counts()`
- Group: `df.groupby('col').agg('other': 'mean')`
- Reset: `df.reset_index(drop=True)`

Key Warnings

- `loc` uses labels; `iloc` uses positions
- After sorting/filtering, indices get scrambled—use `reset_index()`
- Use parentheses around each condition when combining with `&`/`|`
- Mixed types become `object` dtype (slow!)

CS109A: Introduction to Data Science
Lecture 04: k-Nearest Neighbors Regression

Harvard University

Fall 2024

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 04: k-Nearest Neighbors (kNN) Regression
- **Instructor:** Pavlos Protopapas
- **Objective:** Understand statistical modeling fundamentals, the kNN algorithm, model evaluation using MSE and R^2, and the train/validation/test split

Key Summary

This lecture introduces **statistical modeling**—the process of finding mathematical relationships between variables to make predictions. We focus on **k-Nearest Neighbors (kNN)**, an intuitive algorithm that predicts values by looking at "similar" examples in the training data. We also learn how to evaluate models using **Mean Squared Error (MSE)** and **R-squared (R^2)**, and why we must split our data into **training, validation, and test** sets. By the end, you'll understand how to build, evaluate, and compare predictive models.

Contents

1 Key Terminology

Before diving into modeling, let's establish a common vocabulary:

Table 1: *Essential Modeling Terminology*

Term	Also Known As	Description
Response Variable (y)	Target, Dependent Variable, Outcome	The variable we want to predict
Predictor Variables (X)	Features, Independent Variables, Covariates	Variables used to make predictions
Design Matrix (X)	Data Matrix, Feature Matrix	Matrix of all predictors ($n \times p$)
Statistical Model ($\hat{f}$)	Estimator, Predictor	Our approximation of the true relationship
Hyperparameter	Tuning Parameter	Values set by humans before training (e.g., k in kNN)
Loss Function	Cost Function, Objective Function	Measures how wrong the model is
MSE	Mean Squared Error	Average of squared prediction errors
R^2	R-squared, Coefficient of Determination	How much better than the baseline model
Training Set	—	Data used to fit/train the model
Validation Set	Dev Set	Data used to select hyperparameters
Test Set	Holdout Set	Data used ONLY for final evaluation

2 Introduction to Statistical Modeling

2.1 The Prediction Problem

We often want to predict one variable based on others:

- Predict **TikTok views** based on video length, posting time, and past performance
- Predict **movie ratings** based on user history and demographics
- Predict **product sales** based on advertising budget

In this lecture, we'll use the **Advertising Dataset**:

- 200 markets (observations)
- 3 predictors: TV, Radio, Newspaper budgets (in \$1,000s)
- 1 response: Sales (in 1,000 units)

Goal: Build a model to predict Sales given advertising budgets.

2.2 Response vs. Predictor Variables

Not all variables are equal. There's an asymmetry:

Definition:

Response and Predictor Variables

- **Response Variable (y)**: The outcome we're trying to predict
  - Often harder to measure, more important, or influenced by other variables
  - Example: Sales
- **Predictor Variables (X)**: The inputs we use to make predictions
  - Variables we can measure or control
  - Example: TV budget, Radio budget, Newspaper budget

2.3 Mathematical Notation

- y: Response vector of length n (one value per observation)
- X: Design matrix of size $n \times p$ (n observations, p predictors)
- **Convention**: Capital letters = matrices, lowercase = vectors

Example:

Design Matrix Structure With $n = 5$ observations and $p = 3$ predictors:
X **(Design Matrix,** 5×3**):**

TV	Radio	Newspaper
230.1	37.8	69.2
44.5	39.3	45.1
17.2	45.9	69.3
151.5	41.3	58.5
180.8	10.8	58.4

y **(Response Vector,** 5×1**):**

Sales
22.1
10.4
9.3
18.5
12.9

Warning

Shape Matters in Code!

In pandas and sklearn:

- `X.shape` returns `(n, p)` — 2D matrix
- `y.shape` returns `(n,)` (Series) or `(n, 1)` (DataFrame)

The difference between `(n,)` and `(n, 1)` can cause errors. Always check `.shape`!

3 What is a Statistical Model?

3.1 The True Relationship vs. Our Approximation

Analogy:

The Ice Cream Analogy Imagine there exists a **perfect ice cream** recipe—the ideal combination of flavors that no one has ever discovered.

- **True model** f: The perfect, unknown recipe that determines how inputs (ingredients) relate to outputs (taste)
- **Statistical model** $\hat{f}$: Our attempt to recreate that perfect recipe using the ingredients (data) we have

We'll never find the perfect recipe, but we try to get as close as possible!

Mathematically, we assume there exists a true relationship:

$$Y = f(X) + \epsilon$$

- $f(X)$: The systematic, predictable part (the "true" function)
- ϵ: Random noise (measurement error, missing variables, inherent randomness)

Statistical modeling is our attempt to estimate f with $\hat{f}$ using data.

3.2 Two Goals: Inference vs. Prediction

Table 2: *Inference vs. Prediction*

	Inference	**Prediction**
Goal	Understand the *relationship* between X and y	Get accurate *values* for y
Key Question	"How does TV budget *affect* sales?"	"What sales should we *expect* with \$150k TV budget?"
Model Type	Simple, interpretable (e.g., linear regression)	Complex, accurate (e.g., neural networks)
Is $\hat{f}$ interpretable?	**Yes**—we need to understand it	**No**—black box is fine
Analogy	Detective (understanding the crime)	Archer (hitting the target)

This lecture focuses on prediction. We'll cover inference with linear regression later.

4 The Simplest Model: The Mean

Before learning kNN, let's establish the **simplest possible model**—predicting the average:

$$\hat{y} = \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

This model ignores X entirely. No matter what the TV budget is, it always predicts the same value: the average sales (e.g., 12.5).

Key Information

Why Start with the Dumbest Model?

The mean model serves as our **baseline**. Any useful model should beat this baseline!

If your fancy machine learning model can't outperform "just guess the average," something is wrong.

We'll use this baseline to calculate R^2 later.

5 k-Nearest Neighbors (kNN) Algorithm

5.1 The Core Idea

Analogy:

The Doctor's Diagnosis A patient comes in with a stomach ache. The doctor thinks:
"This week, 10 other patients had stomach aches. They all ate from that food truck and got food poisoning. This patient probably has food poisoning too!"
The doctor **found similar past cases** (neighbors) and used their outcomes to make a prediction for the current case.

Definition:

k-Nearest Neighbors (kNN) **kNN** is a non-parametric algorithm that:
1. Finds the k training examples most similar to the query point
2. Averages their y values to make a prediction

Key insight: "Tell me who your neighbors are, and I'll tell you who you are."

5.2 kNN Step by Step (1D Example)

Given: Training data $\{(x_i, y_i)\}$ and a query point x_q

1. **Calculate distances**: Compute $D(x_q, x_i) = |x_q - x_i|$ for all training points
2. **Find k neighbors**: Select the k points with smallest distances
3. **Average**: Compute the prediction as the mean of neighbors' y values:

$$\hat{y}_q = \frac{1}{k} \sum_{i \in \text{Neighbors}_k} y_i$$

Example:

kNN with k=1 **Query**: What's the predicted sales when TV budget is \$150k?
Step 1: Calculate distances from \$150k to all training points
Step 2: Find the closest point (say, \$148k with sales = 18.2)
Step 3: Prediction: $\hat{y} = 18.2$ (just copy the nearest neighbor's value)

5.3 Effect of k on Model Complexity

The choice of k dramatically affects the model:

Warning

The Goldilocks Problem

- **k too small**: Model is too complex, memorizes noise (overfitting)
- **k too large**: Model is too simple, misses patterns (underfitting)

Table 3: *How k Affects kNN*

k Value	Behavior	Problem
k = 1 (small)	Copies nearest neighbor exactly. Very jagged, step-like predictions.	**Overfitting**: Too sensitive to noise
k = 10 (medium)	Averages 10 neighbors. Smoother curve that follows the trend.	Usually a good balance
k = n (large)	Averages ALL data points. Returns the global mean for any query.	**Underfitting**: Ignores local patterns

- **k just right**: Captures the true relationship without the noise

Finding the "just right" k requires model evaluation (next section).

5.4 k is a Hyperparameter

Definition:

Hyperparameter A **hyperparameter** is a value that:

- Is NOT learned from data
- Must be set by the human BEFORE training
- Controls the model's complexity or behavior

In kNN, k is the hyperparameter. We must choose it—the algorithm doesn't learn it.

Question: How do we find the best k? We need to evaluate different models!

6 Model Evaluation

6.1 What Does "Best" Mean?

Before comparing models, we must define "best." For prediction problems:

Best = Lowest prediction error

But how do we measure error?

6.2 Train, Validation, and Test Splits

Important:

The Golden Rule of Model Evaluation You **cannot** evaluate a model on the same data you trained it on!
Why? The model has "seen" that data—it could just memorize the answers.
We need to test on **unseen data** to measure how well the model **generalizes**.

Analogy:

The Exam Analogy

- **Training Set**: Practice problems with answer key. You study from these.
- **Validation Set**: Practice exam. You test yourself to see which study strategy works best.
- **Test Set**: The real final exam. You take it **once** to see your true ability.

If you memorize the practice exam answers instead of learning the material, you'll fail the real exam!

Table 4: *Data Split Purposes*

Set	Purpose	When Used
Training	Fit the model	During model training (kNN stores these points)
Validation	Choose hyperparameters	To compare $k = 1$ vs $k = 10$ vs $k = 70$
Test	Final evaluation	**ONLY ONCE** at the very end

Warning

Data Contamination

"There's a special place in hell for people who use the test set to choose hyperparameters."
—Professor Protopapas

If you peek at the test set while tuning, your final evaluation is **invalid**. The test set must remain **untouched** until the very end.

6.3 Mean Squared Error (MSE)

Definition:

Mean Squared Error **MSE** measures the average squared difference between predictions and actual values:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- y_i: Actual value
- $\hat{y}_i$: Predicted value
- $(y_i - \hat{y}_i)$: Residual (error for one point)

Why square the errors?

- Residuals can be positive or negative
- Without squaring, they might cancel out (e.g., +5 and -5 sum to 0)
- Squaring ensures all errors are positive and penalizes large errors more

Key Information

Why MSE (and not Mean Absolute Error)?
Short answer: MSE has nice mathematical properties (differentiable everywhere).
Deeper answer (covered in Lecture 7): If we assume the noise ϵ follows a Gaussian distribution (which the Central Limit Theorem suggests is often true), then minimizing MSE is mathematically optimal.

6.4 Choosing k Using Validation MSE

1. Split data into train and validation sets
2. For each candidate k (e.g., 1, 3, 5, 10, 20, 50):
   (a) Train kNN on training set
   (b) Compute predictions on validation set
   (c) Calculate validation MSE
3. Choose the k with the **lowest validation MSE**

6.5 R-squared (R^2): Is the Best Model Good Enough?

Analogy:

The Basketball Analogy Suppose Professor Protopapas claims to be the "best basketball player on the teaching team."
Would you sign him to the NBA?
No! Being the best among a small group doesn't mean you're actually good.
Similarly, having the best MSE among your models doesn't mean your model is actually useful.

Definition:

R-squared (R^2) R^2 measures how much better your model is compared to the baseline (mean) model:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\text{MSE}_{\text{model}}}{\text{MSE}_{\text{baseline}}}$$

Interpretation:

- $R^2 = 1$: Perfect predictions ($\hat{y}_i = y_i$ for all i)
- $R^2 = 0$: Model is no better than predicting the mean
- $R^2 < 0$: Model is **worse** than the mean (something is wrong!)

Warning

R^2 Can Be Negative!
Despite the "squared" name, R^2 is NOT the square of anything—it's just a name.
If your model performs worse than the baseline (e.g., due to a bug or overfitting on the wrong data), R^2 will be negative.

7 High-Dimensional kNN

7.1 Multiple Predictors

With more than one predictor, we use **Euclidean distance**:

$$D(\mathbf{x}_q, \mathbf{x}_i) = \sqrt{\sum_{j=1}^{p} (x_{q,j} - x_{i,j})^2}$$

This is just the Pythagorean theorem extended to p dimensions.

7.2 The Curse of Dimensionality

Warning

The Curse of Dimensionality

As the number of dimensions (p) increases:

- Data becomes **sparse**—points spread out in the high-dimensional space
- All points become roughly **equidistant** from each other
- The concept of "nearest neighbor" becomes meaningless

Consequence: kNN struggles in high dimensions unless you have massive amounts of data.

7.3 Feature Scaling

Important:

Scale Your Features! If TV budget is in thousands ($0–$300) but Newspaper budget is in dollars ($0–$300,000), the Newspaper dimension will dominate the distance calculation.

Solution: Standardize all features to have similar scales (covered in sections).

8 Key Takeaways

Key Summary

Summary of Lecture 04

Statistical Modeling Basics

- Response variable (y): What we predict
- Predictor variables (X): What we use to predict
- Goal: Find $\hat{f}$ that approximates the true relationship f

k-Nearest Neighbors

- Non-parametric algorithm: no assumed form for f
- Predicts by averaging the k closest training examples
- k is a hyperparameter: small k = complex/overfit, large k = simple/underfit

Model Evaluation

- Split data into **Train** (fit model), **Validation** (choose hyperparameters), **Test** (final evaluation)
- **MSE**: Average squared error—lower is better
- R^2: How much better than the baseline—higher is better (max 1)

Practical Considerations

- Never use test set for model selection
- In high dimensions, kNN struggles (curse of dimensionality)
- Always scale features when using distance-based methods

8.1 Learning Objectives Checklist

By the end of this lecture, you should be able to:

☐ Define response and predictor variables

☐ Represent data using design matrix X and response vector y

☐ Explain the difference between inference and prediction

☐ Describe kNN as a non-parametric algorithm

☐ Implement kNN in 1D: find neighbors, compute distances, average values

☐ Extend kNN to multiple dimensions using Euclidean distance

☐ Calculate MSE and R^2

☐ Explain the purpose of train/validation/test splits

☐ Recognize the curse of dimensionality and importance of feature scaling

CS109A: Introduction to Data Science
Lecture 05: Linear Regression

Harvard University

Fall 2024

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 05: Linear Regression
- **Instructor:** Pavlos Protopapas
- **Objective:** Master simple and multiple linear regression, understand the normal equation, interpret coefficients, and handle practical issues like scaling and collinearity

Key Summary

This lecture introduces **Linear Regression**—the foundational model for all of machine learning. We start with **Simple Linear Regression** (one predictor), derive the optimal coefficients using calculus, then extend to **Multiple Linear Regression** (many predictors) using matrix notation. We learn to **interpret coefficients**, understand the importance of **feature scaling**, recognize **collinearity** problems, and handle **categorical variables**. Linear regression is the "first principles" model that helps you understand every other ML algorithm.

Contents

1 Key Terminology

Table 1: *Linear Regression Terminology*

Term	**Symbol**	**Description**
Response Variable	Y or y	The outcome we're predicting
Predictor Variable	X or x	Input used for prediction
Intercept	β_0	Value of Y when all Xs are zero
Coefficient/Slope	$\beta_1, \beta_2, ...$	Effect of each predictor on Y
Residual	$r_i = y_i - \hat{y}_i$	Error for one observation
Loss Function	$L(\beta)$	Measures how wrong the model is
MSE	$\frac{1}{n}\sum(y_i - \hat{y}_i)^2$	Mean Squared Error
Normal Equation	$\hat{\beta} = (X^T X)^{-1} X^T y$	Closed-form solution
Design Matrix	X	Matrix of all predictors ($n \times (p+1)$)
Feature Scaling	—	Normalizing predictors to similar scales
Collinearity	—	When predictors are correlated with each other

2 Why Linear Regression?

2.1 The Foundation of All Machine Learning

You might wonder: "Why study something so simple when neural networks exist?"

Key Information

Linear Regression is the Rosetta Stone of ML
Even the most complex models (neural networks, transformers, GPT) follow the same pattern:

1. **Define a model** (structure/hypothesis)
2. **Define a loss function** (measure of error)
3. **Minimize the loss** (find optimal parameters)

Linear regression is the simplest case where you can see this pattern clearly. Master it, and every other algorithm becomes easier to understand.

2.2 Interpretability: The Killer Feature

Unlike kNN ("the neighbors said so"), linear regression tells you **exactly how each variable affects the outcome**.

Example:

kNN vs Linear Regression **Question**: "What happens if we double the TV advertising budget?"
kNN answer: "Uh... let me find similar data points and recalculate..."
Linear Regression answer: "The coefficient is 0.05, so every \$1,000 increase in TV budget increases sales by 50 units. Doubling from \$100k to \$200k should increase sales by approximately 5,000 units."
Linear regression gives you **actionable insights**.

3 Simple Linear Regression (SLR)

3.1 The Model

We assume a **linear relationship** between one predictor X and response Y:

Definition:

Simple Linear Regression

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- β_0: **Intercept** — value of Y when $X = 0$
- β_1: **Slope** — change in Y for a 1-unit change in X
- ϵ: **Error** — everything the model can't explain

Our prediction (without error) is:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

3.2 Finding the Best Line: Loss Function

Among all possible lines, which is "best"? The one that's **closest to all data points**.

3.2.1 Residuals

Definition:

Residual The **residual** for observation i is the vertical distance from the point to the line:

$$r_i = y_i - \hat{y}_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

A good model has small residuals.

3.2.2 Mean Squared Error (MSE)

Definition:

MSE Loss Function

$$L(\beta_0, \beta_1) = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

Warning

Why Square the Residuals?

If we just summed residuals, positive errors ($y > \hat{y}$) and negative errors ($y < \hat{y}$) would cancel out. A model could have huge errors that sum to zero!

Squaring ensures:

1. All errors are positive (no cancellation)

2. Large errors are penalized more heavily ($10^2 = 100$ vs $2^2 = 4$)
3. The function is differentiable everywhere (nice for optimization)

3.3 The Three Steps of Machine Learning

Important:

The Universal ML Recipe

1. **Define the Model**: $\hat{Y} = \beta_0 + \beta_1 X$ (assume a linear relationship)
2. **Define the Loss**: $L = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$ (MSE measures error)
3. **Minimize the Loss**: Find $\hat{\beta}_0, \hat{\beta}_1$ that minimize L

This pattern applies to almost every ML algorithm!

- Neural networks: Same idea, but model is more complex
- ChatGPT: Define transformer model, define cross-entropy loss, minimize with backpropagation

3.4 Minimizing MSE: The Calculus

How do we find the β values that minimize MSE?

Analogy:

Finding the Bottom of a Bowl Imagine MSE as a 3D bowl where:

- x-axis = β_0
- y-axis = β_1
- z-axis = MSE value

The optimal point is at the **bottom of the bowl** where the surface is flat (slope = 0).

Mathematically: Set the **partial derivatives** to zero.

3.4.1 Setting Derivatives to Zero

To find the minimum, we set:

$$\frac{\partial L}{\partial \beta_0} = 0 \quad \text{and} \quad \frac{\partial L}{\partial \beta_1} = 0$$

3.4.2 The Normal Equation (Closed-Form Solution)

Solving these equations gives us:

Simple Linear Regression: Optimal Coefficients

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where $\bar{x}$ and $\bar{y}$ are the means of X and Y.

Key Information

What `.fit()` Actually Does

When you call `model.fit(X, y)` in sklearn, it computes these formulas using your data. That's it—no magic, just algebra!

3.5 Implementation in Python

```python
from sklearn.linear_model import LinearRegression
import pandas as pd

# Load data
df = pd.read_csv('advertising.csv')

# Prepare X and y
X = df[['TV']]          # Double brackets: returns DataFrame (required by sklearn)
y = df['Sales']         # Single brackets: returns Series

# Create and fit model
model = LinearRegression()
model.fit(X, y)         # <-- This runs the normal equation!

# Access the learned parameters
print(f"Intercept (beta_0): {model.intercept_}")
print(f"Slope (beta_1): {model.coef_[0]}")

# Make predictions
prediction = model.predict([[150]])  # Predict sales for TV budget = $150k
```

Listing 1: Simple Linear Regression with sklearn

Warning

Why Double Brackets for X?

sklearn expects X to be a 2D array (matrix), even with one predictor:

- `df['TV']` returns a **Series** with shape `(n,)` — **Error!**
- `df[['TV']]` returns a **DataFrame** with shape `(n, 1)` — **Works!**

4 Multiple Linear Regression (MLR)

4.1 From One to Many Predictors

Real predictions often require multiple inputs:

- Height prediction: weight, biological sex, parents' heights
- Sales prediction: TV budget, radio budget, newspaper budget
- House price: square footage, bedrooms, location, age

Definition:

Multiple Linear Regression

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

Interpretation of β_j: The change in Y for a 1-unit change in X_j, **holding all other predictors constant**.

4.2 Matrix Notation

With many predictors, writing out $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ...$ gets tedious. Matrix notation is cleaner:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

- $\mathbf{Y}$: Response vector $(n \times 1)$
- $\mathbf{X}$: Design matrix $(n \times (p+1))$
- $\boldsymbol{\beta}$: Coefficient vector $((p+1) \times 1)$

4.2.1 The Design Matrix

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

Warning

Why the Column of 1s?

The first column (all 1s) is a mathematical trick to include the intercept β_0 in the matrix multiplication:

$$\begin{pmatrix} 1 & x_1 & x_2 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \beta_0 \cdot 1 + \beta_1 x_1 + \beta_2 x_2$$

Without the 1s column, we'd need to handle β_0 separately.

4.3 The Normal Equation for Multiple Regression

The same logic (set derivatives to zero) gives us the matrix form:

The Normal Equation (Closed-Form Solution)

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

This single formula gives you all optimal coefficients at once!

Key Information

This Only Works for Linear Regression

The closed-form solution is special. For most other models (logistic regression, neural networks, decision trees), no such formula exists. You need iterative methods like gradient descent.

That's why we **love** linear regression—it's mathematically elegant.

4.4 Implementation with Multiple Predictors

```python
# Multiple predictors
X = df[['TV', 'Radio', 'Newspaper']]  # Shape: (n, 3)
y = df['Sales']

model = LinearRegression()
model.fit(X, y)

print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")
# Output might be: [0.046, 0.189, -0.001]
# Interpretation: Radio has strongest effect, Newspaper almost none
```

Listing 2: Multiple Linear Regression

5 Interpreting Coefficients

5.1 What Does a Coefficient Mean?

Definition:

Coefficient Interpretation In the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ...$:
β_j represents the expected change in Y for a **one-unit increase in X_j, holding all other predictors constant**.

Example:

Advertising Example Model: Sales $= 2.94 + 0.046 \cdot \text{TV} + 0.189 \cdot \text{Radio} - 0.001 \cdot \text{Newspaper}$
Interpretations:

- **TV (0.046)**: Each additional \$1,000 in TV budget increases sales by 46 units (holding Radio and Newspaper constant)
- **Radio (0.189)**: Each additional \$1,000 in Radio budget increases sales by 189 units—much larger effect!
- **Newspaper (-0.001)**: Almost zero effect; possibly not worth including

5.2 Feature Importance

To compare which predictors matter most, look at **coefficient magnitudes**:

Warning

Danger: Different Scales!
If TV is in thousands (\$0–300k) and Radio is in hundreds (\$0–50k), comparing raw coefficients is misleading!
A "small" coefficient on a large-scale variable might have more total effect than a "large" coefficient on a small-scale variable.
Solution: **Scale your features** before comparing.

5.3 Coefficient Interpretation Summary

- $\beta = 0$: Predictor has no effect (can be removed)
- $\beta > 0$: Positive relationship (increase $X \rightarrow$ increase Y)
- $\beta < 0$: Negative relationship (increase $X \rightarrow$ decrease Y)
- $|\beta|$ **large**: Strong effect (but check the scale!)

6 Feature Scaling

6.1 Why Scale?

Different predictors often have different units and ranges:

- Age: 0–100
- Income: \$0–\$10,000,000
- Height: 150–200 cm

Without scaling, coefficients can't be directly compared.

6.2 Standardization (Z-Score)

Definition:

Standardization Transform each feature to have mean 0 and standard deviation 1:

$$z = \frac{x - \mu}{\sigma}$$

After standardization, **all features are on the same scale**. A coefficient of 0.5 means "a 1-standard-deviation increase in X produces a 0.5-unit change in Y."

6.3 Min-Max Scaling (Normalization)

Definition:

Min-Max Scaling Transform each feature to the range $[0, 1]$:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

6.4 Does Scaling Affect Predictions?

Key Information

For Linear Regression: No!
Scaling predictors changes the coefficient values but **not the predictions**. If you scale X by 2, the coefficient gets divided by 2—the final prediction is identical.
Why scale then?
1. **Interpretation**: Compare feature importance fairly
2. **Numerical stability**: Prevents very large/small numbers in computations
3. **Other algorithms**: kNN, neural networks **require** scaling

7 Collinearity

7.1 What is Collinearity?

Definition:

Collinearity **Collinearity** (or multicollinearity) occurs when two or more predictors are highly correlated with each other.

Example: Credit card "Limit" and "Rating" are nearly identical—one is probably computed from the other.

7.2 Why is Collinearity a Problem?

Warning

Collinearity Confuses Coefficient Interpretation

If X_1 and X_2 are perfectly correlated:

- When X_1 increases, X_2 also increases
- The model can't tell which one is "causing" the change in Y
- Coefficients become unstable and hard to interpret

Mathematical issue: $(X^T X)^{-1}$ becomes unstable when columns of X are nearly identical.

7.3 Detecting Collinearity

1. **Correlation matrix**: Look for high correlations ($|r| > 0.8$) between predictors
2. **Scatter plot matrix**: Visualize relationships between all pairs
3. **VIF (Variance Inflation Factor)**: Formal measure (covered later)

7.4 What Collinearity Doesn't Break

Key Information

Collinearity is OK for prediction!

If you only care about **predictions** (not interpretation), collinearity doesn't hurt MSE. The predictions will still be accurate.

It only matters when you want to **interpret** which variables are important.

7.5 Handling Collinearity

1. **Remove one of the correlated variables**
2. **Combine them** into a single feature
3. **Use regularization** (Ridge regression—covered later)
4. **Accept it** if you only care about prediction

8 Categorical Predictors

8.1 The Problem with Categories

Linear regression requires **numbers**. But what about:

- Gender: Male/Female
- Color: Red/Blue/Green
- Education: High School/Bachelor's/Master's/PhD

You can't multiply $\beta \times$ "Male"!

8.2 Dummy Variables (One-Hot Encoding)

Definition:

Dummy Variables Convert each category into a **binary (0/1) column**:

Original	IsMale	IsFemale	IsOther
Male	1	0	0
Female	0	1	0
Female	0	1	0
Other	0	0	1

8.3 The Dummy Variable Trap

Warning

Drop One Category!

If you include **all** dummy columns, they sum to 1 (perfect collinearity with the intercept).

Solution: Drop one category (the "reference" or "baseline").

With only IsMale and IsFemale:

- If IsMale = 0 and IsFemale = 0, we know it's "Other"
- The "Other" effect is absorbed into the intercept

8.4 Implementation

```python
import pandas as pd

# Create dummy variables
df_dummies = pd.get_dummies(df, columns=['Gender'], drop_first=True)
# drop_first=True avoids the dummy variable trap

# Now 'Gender_Male', 'Gender_Other' are included (Female is baseline)
X = df_dummies[['Income', 'Age', 'Gender_Male', 'Gender_Other']]
```

Listing 3: Handling Categorical Variables

9 Key Takeaways

Key Summary

Summary of Lecture 05: Linear Regression

The Big Picture

- Linear regression follows the universal ML recipe: Model → Loss → Minimize
- Unlike kNN, it gives **interpretable coefficients**

Simple Linear Regression

- Model: $\hat{Y} = \beta_0 + \beta_1 X$
- Loss: MSE $= \frac{1}{n}\sum(y_i - \hat{y}_i)^2$
- Solution: Set derivatives to zero → Normal equation

Multiple Linear Regression

- Model: $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta}$
- Solution: $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$
- The column of 1s in X handles the intercept

Interpretation

- β_j: Change in Y for 1-unit change in X_j, holding others constant
- Scale features before comparing coefficient magnitudes

Practical Issues

- **Collinearity**: Confuses interpretation (not predictions)
- **Categorical variables**: Use dummy encoding, drop one category
- **Scaling**: Doesn't change predictions, but helps interpretation

9.1 Looking Ahead

Next lectures will cover:

- **Polynomial regression**: What if the relationship isn't linear?
- **Model selection**: How to choose which predictors to include?
- **Regularization**: Ridge and Lasso regression
- **Assumptions of linear regression**: What makes it valid?

Lecture 06: Model Selection and Cross-Validation

CS109A: Introduction to Data Science

Harvard University

■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 06
■ **Instructor:**	Pavlos Protopapas, Kevin Rader, Chris Gumb
■ **Objective:**	Understanding interaction terms, polynomial regression, model selection techniques, and cross-validation for choosing optimal models

Contents

1 Introduction and Motivation

Lecture Overview

This lecture marks a crucial turning point in our machine learning journey. We move beyond simple linear models to tackle more complex relationships in data, while also confronting the central challenge of machine learning: **overfitting**.

Key Topics:

- **Interaction Terms**: Modeling synergy effects between predictors
- **Polynomial Regression**: Capturing nonlinear relationships
- **Overfitting**: Understanding why more complex models aren't always better
- **Model Selection**: Techniques for choosing the best model
- **Cross-Validation**: A robust method for estimating model performance

1.1 The Progression of Model Building

Professor Protopapas emphasizes a fundamental philosophy for approaching any data science project:

The Incremental Approach to Modeling

1. **Start simple**: Begin with the simplest possible model (e.g., linear regression)
2. **Fully understand it**: Interpret coefficients, check assumptions
3. **Ask "Can we do better?"**: Identify limitations of the current model
4. **Add complexity gradually**: Only add features when justified by data

"I dislike with passion" jumping straight to complex models like visual transformers or diffusion models just because you found a tutorial that works. Start with EDA and simple models first!

This approach prevents us from building overly complex models that we don't understand and that may actually perform worse on new data.

1.2 The Mantra of This Lecture

Throughout this lecture, one phrase keeps recurring:

"Too many predictors and collinearity lead to overfitting."

And by the end, we'll extend this to:

"Too many predictors, collinearity, too many interaction terms, and too high polynomial degree lead to overfitting."

2 Understanding Overfitting

2.1 What Is Overfitting?

Before diving into complex models, we need to understand the primary enemy we'll be fighting: **overfitting**.

Definition: Overfitting

Overfitting occurs when a model learns not only the true underlying patterns in the training data but also the **noise** and **outliers**. The model essentially "memorizes" the training data rather than learning generalizable patterns.

Consequence: The model performs excellently on training data but poorly on new, unseen data.

Example: Student Analogy for Overfitting

Imagine overfitting were a student. Which behavior best describes it?

A. Studying just the night before the test

B. **Memorizing every lecture note word for word ← This is overfitting!**

C. Only studying one chapter for all subjects

D. Taking extensive notes but forgetting to understand concepts

The overfitting "student" memorizes everything perfectly but cannot answer questions that are even slightly different from what they memorized. They haven't learned the **concepts**—they've just memorized the **data**.

Example: TA Analogy for Overfitting

If a teaching assistant (TA) were an overfitting model, their grading behavior would be:
"Subtract points for every answer that does not include the word 'overfitting'."
This TA has learned a very specific pattern from limited examples and applies it rigidly, failing to generalize to what good answers actually look like.

2.2 When Does Overfitting Occur?

Overfitting happens when we give our model "too much power." And as the saying goes:

"With great power comes great responsibility."

Factors That Lead to Overfitting

1. **Too many predictors**: High-dimensional feature space
2. **Too high polynomial degree**: Excessively flexible curves
3. **Too many interaction terms**: Modeling spurious relationships
4. **Collinearity**: Redundant information confuses the model

The more complex the model, the more tendency it has to overfit. But if we remove all model power, we get **underfitting**—the model is too simple to capture the true patterns.

2.3 Generalization Error

The ultimate goal of any machine learning model is to perform well on **unseen data**—data the model has never seen during training.

Definition: Generalization Error

Generalization error measures how well a model performs on new, unseen data. It is the error we ultimately care about, not the training error.

Goal of model selection: Find the model that minimizes generalization error (equivalently, the model that minimizes overfitting).

3 Assumptions of Linear Regression Revisited

Before extending linear regression to handle nonlinearity, we need to understand its underlying assumptions. These assumptions come from the two key decisions we made when setting up linear regression:

1. We assumed a **linear relationship** between predictors and response
2. We chose **MSE (Mean Squared Error)** as our loss function

3.1 The Four Key Assumptions

Linear Regression Assumptions from MSE Loss

1. **Linearity**: The relationship between X and Y is linear
   - This is explicit in our model: $Y = \beta_0 + \beta_1 X_1 + \ldots$
2. **Independence**: Errors are independent of each other
   - MSE simply *sums* squared errors
   - If errors were correlated, simple summation would be inappropriate
3. **Homoscedasticity**: Constant variance of errors across all values of X
   - MSE treats all errors *equally* (no weighting)
   - If variance varied, we should weight errors differently
4. **Normality**: Errors are normally distributed
   - Squaring errors connects to normal distribution assumptions
   - This will be explained in more detail in Lecture 9

3.2 Additional Considerations

Two more assumptions are often mentioned (though they don't directly violate linear regression assumptions):

- **Fixed X (No measurement error)**: We assume predictors are measured without error. If there's error in X, Bayesian approaches are needed.
- **No multicollinearity**: Low correlation between predictors. Multicollinearity won't break linear regression but will cause problems with interpretation.

3.3 Diagnosing Assumption Violations: Residual Analysis

How do we know if our assumptions hold? The primary diagnostic tool is **residual analysis**.

Definition: Residuals

The **residual** for observation i is the difference between the actual value and the predicted value:

$$e_i = y_i - \hat{y}_i$$

Residuals represent what our model *couldn't explain*.

3.3.1 Residual Plot Interpretation

We plot residuals (e) against either the predictor (X) or fitted values ($\hat{Y}$):

Reading Residual Plots

Good Model (Assumptions Satisfied):

- Residuals scattered randomly around zero
- No discernible pattern (looks like "white noise")
- Histogram of residuals resembles a bell curve (normal distribution)

Bad Model (Linearity Violated):

- Residuals show a U-shape, S-shape, or other systematic pattern
- This indicates the model is missing a nonlinear trend in the data

Bad Model (Homoscedasticity Violated):

- Residuals show a "fanning" or "funnel" pattern
- Spread of residuals increases (or decreases) with X
- This is called **heteroscedasticity**

Professor Protopapas offers a memorable rule:

*"If you see **any** pattern in your residual plot—even if you imagine dragons flying over castles—that means your assumptions are violated."*

3.3.2 The Histogram Trap

Is a Bell-Shaped Histogram Enough?

If residuals are symmetric (histogram looks normal) but have different spreads at different values of X, the normality assumption may appear satisfied while homoscedasticity is violated.
For example: residuals near $X = 0$ might be small, but residuals for large X might be huge. The marginal histogram (combining all residuals) might still look bell-shaped!
Lesson: Always examine residual plots alongside histograms.

4 Extending Linear Regression: Interaction Terms

Now we begin extending our linear models to handle more complex relationships.

4.1 The Synergy Effect

In reality, the effect of one predictor often depends on the value of another predictor. This is called a **synergy effect** or **interaction effect**.

Example: TV and Radio Advertising

Consider predicting sales based on advertising budgets:
Without considering interaction: If I increase the TV budget by \$1,000, sales increase by β_1 (some fixed amount), regardless of what the radio budget is.
With interaction: If I increase TV budget by \$1,000 *while also having high radio spending*, the effect on sales might be *greater* than if radio spending were low. The two advertising channels might have a synergistic effect.

4.2 Modeling Interactions

To capture interaction effects, we multiply predictors together:

Definition: Interaction Term

For predictors X_1 and X_2, the **interaction term** is $X_1 \times X_2$.
The model becomes:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 \times X_2)$$

Note: The term $X_1 \times X_2$ is **nonlinear** in the predictors (though still linear in the coefficients β).

4.3 Detailed Example: Credit Card Balance

Let's examine the credit card balance dataset, predicting balance from income and student status.

4.3.1 Model Without Interaction

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income} + \beta_2 \times \text{Student}$$

Where Student is a dummy variable: 0 for non-student, 1 for student.

For non-students (Student = 0):

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income}$$

For students (Student = 1):

$$\text{Balance} = (\beta_0 + \beta_2) + \beta_1 \times \text{Income}$$

Interpretation Without Interaction

Both students and non-students have the **same slope** (β_1).
This means: when income increases by \$1,000, balance increases by exactly β_1 dollars—regardless of whether you're a student or not.
Graphically: Two **parallel lines** with different intercepts.

4.3.2 Model With Interaction

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income} + \beta_2 \times \text{Student} + \beta_3 \times (\text{Income} \times \text{Student})$$

For non-students (Student = 0):

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income}$$

For students (Student = 1):

$$\text{Balance} = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) \times \text{Income}$$

Interpretation With Interaction

Now students and non-students have **different slopes**!

- Non-students: slope = β_1
- Students: slope = $\beta_1 + \beta_3$

If $\beta_3 > 0$: Students increase their balance *more* for each additional \$1,000 of income (perhaps they spend more freely).
If $\beta_3 < 0$: Students increase their balance *less* per income increase (perhaps they're more cautious savers).
Graphically: Two lines with different slopes that may cross.

Where Do These Coefficients Come From?

We don't choose $\beta_0, \beta_1, \beta_2, \beta_3$—**the data tells us**!
We fit the model to our training data, and the optimization process finds the coefficient values that minimize MSE.
If $\beta_3 \approx 0$, the data is telling us there's no significant interaction effect.

5 Polynomial Regression

What if the relationship between X and Y is fundamentally curved, not just complicated by interactions?

5.1 When Linear Isn't Enough

Example: Curved Relationships

Imagine plotting your data and seeing this:

- Blue dots follow a clear curved pattern
- The best linear fit (straight line) misses the curve
- Residuals show a systematic U-shaped pattern

What we need is a model that can fit a curved line—a polynomial!

5.2 The Polynomial Model

Definition: Polynomial Regression

A polynomial regression model of degree M is:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \ldots + \beta_M X^M$$

This allows the model to fit curves of varying complexity depending on M.

5.3 The Key Insight: It's Still Linear Regression!

Here's the crucial trick that makes polynomial regression easy:

Polynomial Regression is a Special Case of Multiple Linear Regression

The Trick: Define new "fake" predictors:

$$\begin{aligned}
\tilde{X}_1 &= X \\
\tilde{X}_2 &= X^2 \\
\tilde{X}_3 &= X^3 \\
&\vdots \\
\tilde{X}_M &= X^M
\end{aligned}$$

Now our polynomial model looks like:

$$Y = \beta_0 + \beta_1 \tilde{X}_1 + \beta_2 \tilde{X}_2 + \ldots + \beta_M \tilde{X}_M$$

This is exactly a **multiple linear regression** with M predictors!

Why this matters: We can use the exact same formulas and algorithms we developed for linear regression. The normal equation still works:

$$\hat{\beta} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$$

5.4 Implementation in Python

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Step 1: Create the polynomial features (design matrix)
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)  # Creates columns: 1, X, X^2, X^3

# Step 2: Fit regular linear regression
model = LinearRegression()
model.fit(X_poly, y)

# That's it! Same optimization, same formulas, curvy results.
```

Listing 1: Polynomial Regression in sklearn

5.5 Important Warnings

PolynomialFeatures Creates More Than You Expect

Warning 1: Intercept Column

`PolynomialFeatures` by default creates a column of 1s (the intercept term). If you also use `LinearRegression` with its default `fit_intercept=True`, you'll have **two intercepts**!

Solution: Either:

- Use `PolynomialFeatures(include_bias=False)` and `fit_intercept=True`, OR
- Use `PolynomialFeatures(include_bias=True)` and `fit_intercept=False`

TL;DR: If using polynomial features, set `fit_intercept=False`.

Interaction Terms Are Included!

Warning 2: Multiple Predictors

If you have multiple predictors (e.g., X_1 and X_2) and use `PolynomialFeatures(degree=2)`, you get:

- $1, X_1, X_2$ (degree 0 and 1 terms)
- X_1^2, X_1X_2, X_2^2 (degree 2 terms, including interaction!)

For degree 3, you'd also get $X_1^3, X_1^2X_2, X_1X_2^2, X_2^3$, etc.

This can lead to explosion of features and potential overfitting!

Note: sklearn doesn't have a simple flag to exclude interaction terms. You'd need to manually remove columns.

Feature Scaling is Critical

Warning 3: Numerical Stability

Consider $X = 100$:

- $X^2 = 10,000$
- $X^3 = 1,000,000$
- $X^{10} = 10^{20}$

These vastly different scales cause numerical instability when computing $(\tilde{X}^T\tilde{X})^{-1}$.

Solution: Always standardize your features before polynomial expansion!

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Then apply PolynomialFeatures to X_scaled
```

5.6 Choosing the Polynomial Degree

The degree M is a **hyperparameter**—a choice we must make before training.

- M too low (e.g., $M = 1$): **Underfitting**—model can't capture the curve
- M too high (e.g., $M = 50$): **Overfitting**—model fits every little wiggle and noise
- M just right: Captures the true underlying trend without fitting noise

How do we find the right M? That's the topic of model selection!

6 Model Selection: Finding the Right Balance

Model selection is the process of choosing among different model candidates to find the one that will perform best on new data.

6.1 The Train-Validation-Test Split

The Three Data Splits

1. **Training Set**: Used to *train* the model (find the β coefficients)
2. **Validation Set**: Used to *select* the best model/hyperparameters
3. **Test Set**: Used *once at the very end* to report final model performance

Very Important: The Sacred Test Set

"There's a special place in hell for people who use test data to choose the model."
—Professor Protopapas

The test set must be kept completely separate until final evaluation. Do NOT:

- Use it to tune hyperparameters
- Use it to compare different models
- Look at it during model development

Best practice: Have someone else hold the test set and only evaluate your final model once.

6.2 Model Selection Methods

6.2.1 Exhaustive Search

With J predictors, we could have 2^J possible models (each predictor is either in or out).

Example: Exhaustive Search Complexity

- $J = 3$ predictors $\rightarrow 2^3 = 8$ models (manageable)
- $J = 10$ predictors $\rightarrow 2^{10} = 1,024$ models
- $J = 20$ predictors $\rightarrow 2^{20} = 1,048,576$ models!

Exhaustive search is only practical for very small feature spaces.

6.2.2 Greedy Algorithms

Greedy algorithms make locally optimal choices at each step, without looking at the big picture.

Definition: Forward Selection

Forward Selection is a greedy algorithm for feature selection:

1. Start with model M_0 containing no predictors (just intercept)
2. For each of the J predictors, try adding it to the model
3. Keep the one that reduces validation error the most $\rightarrow$ Model M_1
4. Repeat: try adding each remaining predictor to M_1
5. Keep going until you've tried models with all predictors
6. Select the model with lowest validation error among $M_0, M_1, \ldots, M_J$

Complexity: $O(J^2)$ instead of $O(2^J)$—much faster!

Limitation: May miss optimal combinations (e.g., $X_1 + X_2$ might be best, but if X_3 alone beats X_1 alone, we'd never try $X_1 + X_2$).

Other greedy approaches include:

- **Backward Elimination**: Start with all predictors, remove the least helpful one at each step
- **Stepwise Selection**: Combination of forward and backward

6.3 Hyperparameter Tuning with Validation

For polynomial regression, we tune the degree M using the validation set:

1. For each candidate degree $M \in \{1, 2, 3, \ldots, 10\}$:
   (a) Train the model on the training set
   (b) Compute MSE on the validation set
2. Plot validation MSE vs. degree M
3. Select the M with lowest validation MSE

6.3.1 The U-Shaped Curve

Training Error vs. Validation Error

Training Error (MSE on training data):

- Decreases monotonically as model complexity increases
- More complex model can always fit training data better (or at least as well)
- Can reach zero with enough complexity (just memorize the data!)

Validation Error (MSE on validation data):

- Initially decreases as complexity captures true patterns
- Reaches a minimum at the "sweet spot"
- Then **increases** as model starts fitting noise (overfitting)

The validation error curve is typically **U-shaped**. We choose the complexity at the bottom of the U.

7 Cross-Validation: A Robust Approach

7.1 The Problem with a Single Validation Set

Using a single validation set has a critical flaw:

Example: Overfitting to the Validation Set

Suppose the true underlying relationship is cubic (degree 3).
But by chance, our randomly chosen validation points happen to lie almost perfectly on a straight line!
When we compare models:

- Degree 1 model: Low validation error (lucky fit)
- Degree 3 model: Higher validation error (validation points don't follow cubic well)

We'd incorrectly choose degree 1!
The problem: We avoided overfitting to training data, but ended up **overfitting to the validation data**.

7.2 The Solution: K-Fold Cross-Validation

The remedy is to validate on **multiple different splits** and average the results.

Definition: K-Fold Cross-Validation

K-Fold Cross-Validation procedure:

1. Set aside the test set (never touch it during CV)
2. Divide the remaining data into K equal "folds" (typically $K = 5$ or $K = 10$)
3. For $i = 1$ to K:
   (a) Use fold i as the validation set

(b) Use all other $K-1$ folds as the training set

(c) Train the model and compute validation error MSE_i

4. Compute the average: $CV = \frac{1}{K}\sum_{i=1}^{K} MSE_i$

This CV score is a more robust estimate of how the model will perform on new data.

7.2.1 Visual Representation of 5-Fold CV

Iteration	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	**Val**	Train	Train	Train	Train
2	Train	**Val**	Train	Train	Train
3	Train	Train	**Val**	Train	Train
4	Train	Train	Train	**Val**	Train
5	Train	Train	Train	Train	**Val**

Each data point gets to be in the validation set exactly once!

7.3 Using Cross-Validation for Model Selection

To select the best hyperparameter (e.g., polynomial degree M):

1. For each candidate $M \in \{1, 2, 3, \ldots, 10\}$:
   (a) Perform K-fold CV
   (b) Record the CV score (average validation MSE)
2. Select the M with the lowest CV score
3. (Optional) Retrain on all training data with chosen M
4. Evaluate final performance on test set

7.4 Leave-One-Out Cross-Validation (LOOCV)

An extreme case of K-fold CV where $K = N$ (number of data points):

- Each iteration uses just 1 point for validation, $N-1$ for training
- Repeated N times
- **Pros**: Very low bias in estimate
- **Cons**: Computationally expensive (train N models!)

In practice, $K = 5$ or $K = 10$ provides a good balance between bias and computational cost.

7.5 Implementation: The Negative MSE Trick

sklearn Uses Negative MSE

sklearn's cross-validation functions are designed to **maximize** a scoring metric (like accuracy). But we want to **minimize** MSE!

Solution: Use `scoring='neg_mean_squared_error'`

The function returns *negative* MSE values. To get actual MSE:

```
cv_results = cross_validate(model, X, y, cv=5,
                            scoring='neg_mean_squared_error')
actual_mse = -cv_results['test_score'].mean()
```

```
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
import numpy as np

# Find best polynomial degree using CV
degrees = range(1, 11)
cv_scores = []

for degree in degrees:
    # Create pipeline: Scale -> Polynomial -> Linear Regression
    model = make_pipeline(
        StandardScaler(),
        PolynomialFeatures(degree=degree, include_bias=False),
        LinearRegression()
    )

    # 5-fold cross-validation
    cv_results = cross_validate(
        model, X, y, cv=5,
        scoring='neg_mean_squared_error',
        return_train_score=True
    )

    # Convert to positive MSE and store
    cv_scores.append(-cv_results['test_score'].mean())

# Find best degree
best_degree = degrees[np.argmin(cv_scores)]
print(f"Best polynomial degree: {best_degree}")
```

Listing 2: Complete Cross-Validation Example

8 Interpreting Models: Beyond Numbers

Even when a model has good MSE or R^2, we must interpret it to ensure it makes sense.

Example: When Numbers Lie

Case 1: Model for TV budget (X) vs. Sales (Y):

$$Y = -0.05X + 6.2$$

Problem: Negative slope means more TV spending leads to *less* sales. Does that make sense? Probably not! Either:

- The data is wrong
- The model is misspecified
- There's some confounding factor we're missing

Case 2: Another model:

$$Y = 0.02X - 0.5$$

Problem: Negative intercept means when $X = 0$ (no TV spending), sales are *negative*. That's impossible!

Lesson: Always sanity-check your model coefficients against domain knowledge.

The Complete Model Evaluation Checklist

1. Check quantitative metrics (MSE, R^2)
2. Examine residual plots for assumption violations
3. Interpret coefficients—do they make intuitive sense?
4. Consider the domain context—is the model telling a plausible story?
5. Validate on held-out data to ensure generalization

9 Dealing with Categorical Variables (Reminder)

9.1 Two Categories: Dummy Variables

For a categorical variable with 2 categories (e.g., Student: Yes/No):

- Create one dummy variable: 0 = No, 1 = Yes

9.2 More Than Two Categories: One-Hot Encoding

Example: Encoding Ethnicity

Variable: Ethnicity with categories Asian, Caucasian, African-American

Wrong approach: Code as 0, 1, 2

- This implies ordering ($0 < 1 < 2$)

- Implies equal "distances" between categories
- Neither makes sense for categorical data!

Correct approach: One-hot encoding

- Create 3 dummy variables: `Is_Asian`, `Is_Caucasian`, `Is_AfricanAmerican`
- Each observation has exactly one 1 and two 0s

But wait: We can drop one column! If we know someone is not Asian and not Caucasian, they must be African-American.

So we use only 2 dummy variables, and the third category becomes the "baseline."

Interpreting Coefficients with Dummy Variables

Model: $\text{Balance} = \beta_0 + \beta_1 \times \text{Is\_Asian} + \beta_2 \times \text{Is\_Caucasian}$

Interpretations:

- β_0: Average balance for African-Americans (the baseline)
- $\beta_0 + \beta_1$: Average balance for Asians
- $\beta_0 + \beta_2$: Average balance for Caucasians
- β_1: Difference in balance between Asians and African-Americans
- β_2: Difference in balance between Caucasians and African-Americans

10 Quick Reference Summary

Lecture 06 Quick Reference Card

1. Overfitting

- Model memorizes training data including noise
- Low training error, high test/validation error
- Caused by: too many predictors, high polynomial degree, too many interactions
- Solution: Model selection with validation/cross-validation

2. Interaction Terms

- Model: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 \times X_2)$
- Captures synergy effects between predictors
- Changes the slope of one variable based on another

3. Polynomial Regression

- Model: $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_M X^M$
- Special case of multiple linear regression (linear in β!)
- Warnings: Scale features, watch for intercept duplication

4. Data Splits

- **Training**: Fit model parameters (β)
- **Validation**: Choose hyperparameters/model
- **Test**: Final performance report (use ONCE!)

5. K-Fold Cross-Validation

- Split data into K folds, rotate validation fold
- Average K validation scores for robust estimate
- Avoids overfitting to a single validation split
- Typical K: 5 or 10

11 Common Questions and Answers

Q: How can polynomial regression be "linear" when it fits curves?

A: It's linear *in the coefficients*. The model $Y = \beta_0 + \beta_1 X + \beta_2 X^2$ is nonlinear in X (the graph is curved), but it's a linear combination of the β terms. We can treat X^2 as a new variable $\tilde{X}$, and then it's just $Y = \beta_0 + \beta_1 X + \beta_2 \tilde{X}$—standard multiple linear regression!

Q: What's the difference between validation and test sets?

A: Purpose!

- **Validation**: Used repeatedly during development to compare models and tune hyperparameters
- **Test**: Used exactly once at the end to report final performance

Think of validation as "practice exams" and test as the "final exam."

Q: How do I choose K for K-fold CV?

A: There's no perfect answer, but $K = 5$ or $K = 10$ are standard choices. They balance:

- Bias (larger K = less bias, since training sets are larger)
- Variance (larger K = more variance, since validation sets are smaller)
- Computational cost (larger K = more model fits)

Q: Should I always standardize my features?

A: Not always required, but usually a good idea:

- Required for: Polynomial regression, regularization, KNN, SVM, neural networks
- Optional for: Simple/multiple linear regression (doesn't affect predictions, only coefficient interpretation)

When in doubt, standardize!

Q: My model has high R^2 but coefficients don't make sense. What's wrong?

A: Several possibilities:

- Multicollinearity between predictors
- Overfitting to noise
- Data errors or preprocessing issues
- Confounding variables not in the model

Always combine quantitative metrics with qualitative interpretation!

12 Looking Ahead

In the next lecture (Lecture 07), we will explore:

- **Regularization**: A powerful technique to prevent overfitting by penalizing large coefficients
- **Ridge Regression**: L2 regularization
- **Lasso Regression**: L1 regularization (also performs feature selection!)
- **Bias-Variance Tradeoff**: The fundamental tension in machine learning

These techniques give us another tool (beyond cross-validation) to combat overfitting.

Lecture 07: Bias-Variance Tradeoff and Regularization

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 07
- **Instructor:** Pavlos Protopapas, Kevin Rader, Chris Gumb
- **Objective:** Understanding the bias-variance tradeoff, diagnosing overfitting through coefficient analysis, and learning regularization techniques (Ridge and Lasso) to combat overfitting

Contents

1 Introduction and Motivation

Lecture Overview

This lecture addresses one of the most fundamental concepts in machine learning: the **bias-variance tradeoff**. We'll understand why models fail, how to diagnose the problem, and introduce powerful techniques called **regularization** to fix it.

Key Topics:

- **Sources of Error**: Irreducible vs. reducible error
- **Bias-Variance Tradeoff**: The fundamental tension in model complexity
- **Diagnosing Overfitting**: Looking at coefficient magnitudes
- **Ridge Regression (L2)**: Penalizing squared coefficients
- **Lasso Regression (L1)**: Penalizing absolute coefficients (with feature selection!)
- **Hyperparameter Tuning**: Finding optimal λ using cross-validation

1.1 The Big Picture

Our ultimate goal is **generalization**—building models that perform well on *new, unseen data*, not just the training data we happened to collect.

We've learned that:

- Too simple models **underfit**: They can't capture the true patterns
- Too complex models **overfit**: They memorize noise instead of learning patterns

Today we formalize this intuition mathematically and introduce techniques to navigate between these extremes.

2 Sources of Prediction Error

When our model makes errors on test data, those errors can come from different sources.

2.1 Irreducible Error (Aleatoric Error)

Definition: Irreducible Error

Irreducible error is the inherent noise in the data that cannot be eliminated no matter how good our model is. This is also called **aleatoric error**.

This error exists because:

- Measurements have inherent randomness
- There are unmeasured variables affecting the outcome
- The relationship itself may have stochastic components

Example: Irreducible Error Analogy

Imagine recording audio with the world's best microphone. Even with perfect equipment, you'll still pick up ambient noise—air molecules vibrating, electronic interference, etc.
No matter how much you improve the microphone (model), you can't eliminate environmental noise (irreducible error).
Acceptance: We acknowledge this error exists and focus on what we *can* control.

2.2 Reducible Error

The error we *can* control comes from our choice of model. This **reducible error** decomposes into two components:

1. **Bias**: Error from wrong assumptions in the model
2. **Variance**: Error from sensitivity to training data fluctuations

3 The Bias-Variance Tradeoff

3.1 Understanding Bias

Definition: Bias

Bias measures how far off the model's average prediction is from the true value.

- **High Bias**: Model consistently misses the target (inaccurate)
- **Low Bias**: Model's predictions are centered around the truth

High bias typically occurs when the model is **too simple** to capture the underlying patterns.

Example: High Bias: The Linear Model on Curved Data

Imagine the true relationship is curved (like a parabola), but you fit a straight line.
No matter how many times you collect new data and refit:

- Every fitted line will miss the curve
- The average of all fitted lines still misses the truth
- This systematic error is **bias**

The model is fundamentally incapable of capturing the true pattern—it's **underfitting**.

3.2 Understanding Variance

Definition: Variance

Variance measures how much the model's predictions change when trained on different datasets.

- **High Variance**: Predictions swing wildly with different training data
- **Low Variance**: Predictions are stable regardless of training data

High variance typically occurs when the model is **too complex** and learns noise specific to each training set.

Example: High Variance: The Spaghetti Plot

Professor Protopapas demonstrates this beautifully with a simulation:

Experiment: Take 2,000 different random samples from the same population. For each sample, fit a model. Plot all 2,000 fitted curves.

Linear Model (Low Variance): All 2,000 lines cluster tightly together—very stable predictions across different training sets.

Degree-10 Polynomial (High Variance): The 2,000 curves look like "spaghetti noodles"—wildly different predictions for each training set. The model is chasing the noise in each particular sample.

3.3 The Tradeoff Visualized

The Fundamental Tradeoff

As model complexity increases:

- **Bias decreases**: More complex models can fit more patterns
- **Variance increases**: More complex models are more sensitive to noise

As model complexity decreases:

- **Bias increases**: Simpler models miss true patterns
- **Variance decreases**: Simpler models are more stable

Total Error = Bias^2 + Variance + Irreducible Error

This creates a U-shaped curve when plotted against complexity. Our goal is to find the **sweet spot** at the bottom of the U.

3.4 The Dartboard Analogy

Think of model predictions like throwing darts at a target:

	Low Variance	**High Variance**
High Bias	Darts clustered together, but away from bullseye (Underfitting)	Darts scattered everywhere, missing the bullseye (Worst case)
Low Bias	Darts clustered tightly around the bullseye (**Ideal model!**)	Darts scattered around the bullseye (Overfitting)

Table 1: *The four scenarios of bias and variance*

4 Diagnosing Overfitting: Coefficient Analysis

How can we tell if our model is overfitting? Look at the **coefficients**!

4.1 The Key Insight

Overfitting Symptom: Exploding Coefficients

When a model overfits:

- Coefficients (β values) become **extremely large**
- Coefficient values are **unstable**—they vary wildly across different training sets

This happens because the model is trying to fit every tiny wiggle in the training data, requiring extreme parameter values.

Example: Coefficient Distributions: Linear vs. Polynomial

From the 2,000-model simulation:

Linear Model:

- β_0 values range from roughly 0.0 to 1.25
- β_1 values similarly bounded
- Narrow violin plots (low variance in coefficients)

Degree-10 Polynomial:

- Some coefficients ($\beta_5, \beta_8, \beta_9$) reach values on the order of 10^9 (one billion!)
- Extremely wide violin plots (high variance in coefficients)
- The y-axis label shows "1e9"—coefficients are astronomically large

4.2 The Solution Preview

This observation leads directly to our solution:

If overfitting = large coefficients,

then preventing overfitting = keeping coefficients small!

This is the core idea behind **regularization**.

5 Regularization: The Core Idea

5.1 Penalizing Complexity

Definition: Regularization

Regularization modifies the loss function to penalize model complexity, encouraging simpler models that generalize better.

Instead of minimizing just MSE, we minimize:

$$\mathcal{L}_{\text{regularized}} = \underbrace{\text{MSE}}_{\text{Fit the data}} + \underbrace{\lambda \cdot \text{Penalty}}_{\text{Keep model simple}}$$

The model must now balance two competing objectives:

1. Fit the training data well (minimize MSE)
2. Keep coefficients small (minimize penalty)

Example: Game Time: How to Discourage Large Coefficients?

Professor Protopapas poses this question to the class:

Options:

A. Divide all model parameters by large numbers

B. Make sure the causal relation between predictors and response is true

C. Discard any model with parameter values larger than one

D. Penalize the model with a penalty proportional to parameter values

Answer: D!

Why not A? Dividing just scales the data—nothing fundamentally changes.

Why not C? Discarding models creates bias and arbitrary cutoffs.

The right approach: Add a term to the loss function that grows when coefficients grow.

5.2 The Regularization Parameter λ

The hyperparameter λ (lambda) controls the **strength** of regularization:

Understanding λ

- $\lambda = 0$: No regularization. This is just ordinary linear regression. Risk of overfitting if model is complex.
- λ **small**: Mild regularization. Coefficients are somewhat constrained.
- λ **large**: Strong regularization. Coefficients are heavily constrained, pushed toward zero.
- $\lambda \to \infty$: Extreme regularization. All coefficients become exactly zero. The model predicts just the mean (severe underfitting).

We need to find the λ that balances bias and variance optimally.

6 Ridge Regression (L2 Regularization)

6.1 The Ridge Loss Function

Definition: Ridge Regression

Ridge Regression uses the **L2 norm** (sum of squared coefficients) as the penalty:

$$\mathcal{L}_{\text{Ridge}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Equivalently:

$$\mathcal{L}_{\text{Ridge}} = \text{MSE} + \lambda \|\boldsymbol{\beta}\|_2^2$$

where $\|\boldsymbol{\beta}\|_2^2 = \beta_1^2 + \beta_2^2 + \ldots + \beta_p^2$

Important: Don't Regularize the Intercept!

Notice the penalty sums from $j = 1$ to p, **not** including β_0 (the intercept).

Why? The intercept β_0 is just an overall offset—it doesn't relate to any predictor's influence. Overfitting comes from being too sensitive to predictors, not from the baseline level.

Always exclude β_0 from regularization.

6.2 Properties of Ridge Regression

Ridge Regression Characteristics

How it shrinks coefficients:

- Shrinks all coefficients **toward zero**
- But coefficients never become **exactly** zero
- Larger coefficients are penalized more heavily (due to squaring)

Advantages:

- **Closed-form solution**: Very fast to compute

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (X^T X + \lambda I)^{-1} X^T y$$

- Excellent for **multicollinearity**: Stabilizes estimates when predictors are correlated
- Keeps all predictors in the model (useful when you believe all matter)

When to use:

- When you believe all predictors contribute somewhat
- When predictors are highly correlated
- When you need fast computation

7 Lasso Regression (L1 Regularization)

7.1 The Lasso Loss Function

Definition: Lasso Regression

Lasso Regression (Least Absolute Shrinkage and Selection Operator) uses the **L1 norm** (sum of absolute values) as the penalty:

$$\mathcal{L}_{\text{Lasso}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Equivalently:

$$\mathcal{L}_{\text{Lasso}} = \text{MSE} + \lambda \|\boldsymbol{\beta}\|_1$$

where $\|\boldsymbol{\beta}\|_1 = |\beta_1| + |\beta_2| + \ldots + |\beta_p|$

7.2 Properties of Lasso Regression

Lasso Regression Characteristics

How it shrinks coefficients:

- Can shrink coefficients to **exactly zero**
- Effectively **removes** predictors from the model
- Performs automatic **feature selection**

Advantages:

- **Sparse solutions**: Produces interpretable models with fewer predictors
- **Feature selection**: Identifies which predictors actually matter
- Useful when you suspect only a few predictors are truly important

Disadvantages:

- **No closed-form solution**: Must use numerical optimization (slower)
- With correlated predictors, tends to pick one arbitrarily and zero others

When to use:

- When you have many predictors and suspect few are truly relevant
- When you want an interpretable model
- When you need automatic feature selection

7.3 Why Lasso Produces Zeros (Intuition)

Example: Why L1 Creates Sparsity

The geometric intuition involves the shape of the constraint regions:

Ridge (L2): The constraint $\sum \beta_j^2 \leq c$ forms a **circle** (or sphere in higher dimensions). The MSE contours typically intersect this circle somewhere on the smooth boundary—rarely exactly at an axis.

Lasso (L1): The constraint $\sum |\beta_j| \leq c$ forms a **diamond** (or cross-polytope). The corners of the diamond lie on the axes. The MSE contours are much more likely to hit a corner, where some coefficient is exactly zero.

This is why Lasso naturally produces sparse solutions!

Aspect	Ridge (L2)	Lasso (L1)
Penalty term	$\sum \beta_j^2$	$\sum \lvert\beta_j\rvert$
Coefficient shrinkage	Toward zero, never exactly zero	Can be exactly zero
Feature selection	No (keeps all predictors)	Yes (automatic)
Solution type	Closed-form (analytical)	Numerical solver
Computation speed	Fast	Slower
Multicollinearity	Handles well	May pick one predictor arbitrarily
Interpretability	All predictors remain	Sparse, easier to interpret
Best when	All predictors matter	Few predictors matter

Table 2: *Ridge vs. Lasso comparison*

8 Ridge vs. Lasso: Comparison

Which to Choose?

The honest answer: Try both and use cross-validation to compare!

General guidelines:

- If you have hundreds of predictors but suspect only a handful matter → **Lasso**
- If you believe all predictors contribute at least a little → **Ridge**
- If predictors are highly correlated → **Ridge** (more stable)
- If you need interpretability and feature selection → **Lasso**

There's also **Elastic Net** which combines both penalties—a topic for another day!

9 Finding Optimal λ: The Complete Procedure

λ is a **hyperparameter**—it's not learned from data but must be set by us. We find it through validation or cross-validation.

Very Important: Hyperparameter Tuning Rules

- **NEVER** tune on training data (model will choose $\lambda = 0$)
- **NEVER** tune on test data (that's cheating—information leakage)
- **ALWAYS** use validation set or cross-validation

9.1 Method 1: Single Validation Set

1. **Split data**: Training / Validation / Test
2. **Choose λ candidates**: Select a range to search
   - Typically: $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100\}$
   - Use logarithmic spacing (orders of magnitude)
3. **For each** λ: Train model on training set, find $\boldsymbol{\beta}_\lambda$
   - Ridge: Use closed-form formula
   - Lasso: Use numerical solver

4. **Evaluate on validation set**: Compute MSE **without** the penalty term!

Critical Point

When evaluating models, compute **pure MSE only**—don't include the $\lambda \cdot$ penalty term.
Why? The penalty was a training tool to prevent overfitting. For evaluation, we care about actual prediction accuracy (how close to true values), not about coefficient sizes.

5. **Select best** λ^*: Choose the λ with lowest validation MSE
6. **Refit (recommended)**: Combine training + validation data, retrain with λ^*
   - Model selection is done—no need to keep validation separate
   - More training data = better final model
7. **Final evaluation**: Report MSE on test set (use only once!)

9.2 Method 2: K-Fold Cross-Validation

A more robust approach that reduces dependence on a single validation split:

1. **Split data**: Training Pool (for CV) / Test
2. **Choose** λ **candidates**: Same as before
3. **Divide training pool into K folds**: Typically $K = 5$ or $K = 10$
4. **For each** λ**, run K iterations**:
   - Iteration 1: Fold 1 = validation, Folds 2-K = training
   - Iteration 2: Fold 2 = validation, Folds 1,3-K = training
   - ... and so on ...
   - Record validation MSE for each iteration
5. **Average MSEs**: For each λ, compute mean of K validation MSEs
6. **Select best** λ^*: Choose λ with lowest **average** validation MSE
7. **Refit on all training data**: Train final model using entire training pool with λ^*
   - Don't use one of the K models—retrain on all data
   - This ensures maximum information for final model
8. **Final evaluation**: Report MSE on test set

Example: Why Refit After Cross-Validation?

After K-fold CV, you have K different models (one per fold). Which do you use?
Answer: None of them! Each was trained on only $(K-1)/K$ of the data.
Instead, now that you've found λ^*, retrain a **new model** on the **entire** training pool. This final model benefits from all available training data.

9.3 Practical Implementation

```
from sklearn.linear_model import Ridge, Lasso, RidgeCV, LassoCV
from sklearn.model_selection import cross_val_score
import numpy as np
```

```python

# Define lambda values to search (sklearn calls it 'alpha')
alphas = np.logspace(-5, 2, 50)  # 50 values from 10^-5 to 10^2

# Method 1: Manual cross-validation
ridge = Ridge(alpha=1.0)
cv_scores = cross_val_score(ridge, X_train, y_train,
                            cv=5,
                            scoring='neg_mean_squared_error')
mean_mse = -cv_scores.mean()

# Method 2: Built-in CV (recommended)
# RidgeCV automatically finds best alpha
ridge_cv = RidgeCV(alphas=alphas, cv=5)
ridge_cv.fit(X_train, y_train)
print(f"Best alpha: {ridge_cv.alpha_}")

# Similarly for Lasso
lasso_cv = LassoCV(alphas=alphas, cv=5)
lasso_cv.fit(X_train, y_train)
print(f"Best alpha: {lasso_cv.alpha_}")

# Final evaluation on test set
test_mse = mean_squared_error(y_test, ridge_cv.predict(X_test))
```

Listing 1: Ridge and Lasso with Cross-Validation in sklearn

Edge Cases in λ Selection

What if the optimal λ is at the edge of your search range?
For example, if your lowest validation MSE occurs at $\lambda = 100$ (your maximum):
Problem: The true optimum might be at $\lambda = 1000$ or higher!
Solution: Expand your search range and try again. The optimal λ should be in the "middle" of the U-shaped curve, not at the boundary.
If optimal is at $\lambda = 10^{-5}$ (your minimum), try smaller values.

10 Common Questions and Answers

Q: Why don't we include the regularization term when evaluating on validation?

A: The penalty term was a *training tool* to prevent overfitting—like training wheels on a bike. When we evaluate the model's true performance, we care about prediction accuracy (how close predictions are to actual values), not how "simple" the model is. MSE measures prediction quality; the penalty doesn't.

Q: Should I use the same λ search range for all problems?

A: No! The appropriate range depends on your data scale and problem. Start with a wide range (e.g., 10^{-5} to 10^{5}), visualize the results, and narrow down. If the optimum is at an edge, expand the range.

Q: Can I tune K in K-fold CV?

A: Technically yes, but it's usually not worth it. K=5 or K=10 work well in practice. The difference is rarely significant, and tuning K with another CV creates complexity. Just pick 5 or 10 and move on.

Q: What about Elastic Net?

A: Elastic Net combines Ridge and Lasso penalties:

$$\mathcal{L}_{\text{ElasticNet}} = \text{MSE} + \lambda_1 \sum |\beta_j| + \lambda_2 \sum \beta_j^2$$

It gets the best of both worlds: feature selection from Lasso and stability from Ridge. But now you have *two* hyperparameters to tune!

Q: How much of data science is "art" vs. "science"?

A: Professor Protopapas says roughly 70% science, 30% art. Not everything has proofs. With hyperparameters, you develop intuition over time. Start with reasonable defaults, use cross-validation, and accept that "good enough" is often the goal. You can't search infinite parameter spaces.

11 Quick Reference Summary

Lecture 07 Quick Reference Card

1. Error Decomposition

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

- High Bias = Underfitting (too simple)
- High Variance = Overfitting (too complex)

2. Overfitting Diagnosis

Overfitting $\Rightarrow$ Large, unstable coefficients
Solution: Penalize large coefficients!

3. Regularization Formula

$$\mathcal{L} = \text{MSE} + \lambda \cdot \text{Penalty}$$

- Ridge (L2): Penalty $= \sum \beta_j^2$ (shrinks toward zero)
- Lasso (L1): Penalty $= \sum |\beta_j|$ (can make zeros)

Don't regularize β_0!

4. Finding λ^*

1. Choose range: $\{10^{-5}, \ldots, 10^2\}$
2. For each λ: train, compute validation MSE (no penalty!)
3. Choose λ^* with minimum validation MSE
4. Refit on train+validation with λ^*
5. Report on test (once!)

5. Ridge vs. Lasso

- Ridge: Fast, keeps all predictors, good for multicollinearity
- Lasso: Feature selection, sparse models, slower
- When unsure: Try both, compare with CV

12 Looking Ahead

With regularization in our toolkit, we now have powerful techniques to:

- Prevent overfitting without reducing model complexity
- Automatically select important features (Lasso)

- Handle multicollinearity (Ridge)

In upcoming lectures, we'll move beyond regression to **classification** problems, where we predict categories rather than continuous values. Many of the same principles—bias-variance tradeoff, regularization, cross-validation—will apply!

Lecture 08: Statistical Inference for Regression

CS109A: Introduction to Data Science

Harvard University

■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 08

■ **Instructor:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Objective:** Understanding uncertainty in regression coefficients through bootstrapping, building confidence intervals, evaluating predictor significance using t-tests and p-values, and distinguishing confidence intervals from prediction intervals

Contents

1 Introduction: Why Do We Need Inference?

Lecture Overview

So far, we've learned how to *fit* models and *predict* outcomes. But fitting a model gives us just one estimate—one set of coefficients based on one sample of data. How certain can we be about these numbers?

This lecture answers a critical question: **"How much should we trust our model?"**

Key Topics:

- **Accuracy of Estimates**: How precise are our $\hat{\beta}$ values?
- **Bootstrapping**: Simulating "parallel universes" to measure uncertainty
- **Confidence Intervals**: A range where the true β likely falls
- **Feature Importance**: Which predictors actually matter?
- **Statistical Significance**: Is the effect real or just random noise?
- **Prediction Intervals**: How uncertain are our predictions $\hat{y}$?

1.1 The Consultant Scenario

Professor Protopapas sets up a vivid scenario:

Example: The $10

Imagine you're a consultant who built a model for advertising:

$$\hat{y} = 1.01x + 0.05$$

Where x = TV advertising budget (in thousands), y = sales (in thousands).

Your interpretation: "For every $1,000 spent on TV ads, sales increase by $1,010. Net profit: $10 per $1,000 invested."

The question: If you go to your boss and ask for $10,000 for your analysis, will they pay?

The doubt: That coefficient 1.01 came from *one* sample of data. What if you had collected data on a different day? You might have gotten 1.03, or 0.98, or something completely different!

The core issue: How do we *quantify* and *communicate* our uncertainty?

1.2 Sources of Uncertainty

Before measuring uncertainty, let's understand where it comes from:

Definition: Two Types of Error

1. Irreducible Error (Aleatoric Error, ϵ)

- Inherent randomness in the system
- Even with perfect model, there's noise we can't eliminate
- Example: Same ad budget on different days yields different sales due to weather, competitor

actions, random human behavior

2. Reducible Error

- **Model misspecification**: We assumed linear but reality is curved
- **Limited samples**: We only observed one "realization" of reality
- Can be reduced with better models or more data

For this lecture: We bundle everything into ϵ (epsilon)—the error term that makes our $\hat{\beta}$ uncertain.

2 The Thought Experiment: Parallel Universes

2.1 What If We Could Repeat the Experiment?

To understand how much $\hat{\beta}$ varies, imagine this scenario:

The Parallel Universe Thought Experiment

Scenario: We know the true relationship $y = f(x) + \epsilon$. But due to error ϵ, each measurement is slightly different from the true value.

Process:

1. **Universe 1**: Collect data (with random error). Fit model. Get $\hat{\beta}^{(1)}$.
2. **Universe 2**: Collect new data (different random error). Fit model. Get $\hat{\beta}^{(2)}$.
3. **Universe 3**: Collect new data. Get $\hat{\beta}^{(3)}$.
4. ... repeat 100 times ...

Result: We get 100 different $\hat{\beta}$ values. We can plot their histogram!

Insight: If the histogram is **narrow**, our estimate is precise. If **wide**, it's uncertain.

Example: Visualizing the Spaghetti Plot

If we plot all 100 fitted regression lines from our parallel universes, we get a "spaghetti plot"—many lines clustered together.

- **Tight spaghetti**: All lines are similar → Low variance → Confident estimate
- **Wild spaghetti**: Lines spread everywhere → High variance → Uncertain estimate

(This is exactly what we saw in Lecture 07 when discussing bias-variance!)

2.2 The Problem: We Can't Actually Visit Parallel Universes

This thought experiment is beautiful, but impossible in practice. We only have **one** dataset. We can't go back in time and recollect data under different random conditions.

Solution: **Bootstrapping**—a clever trick to simulate parallel universes using only the data we have!

3 Bootstrapping: Creating "Parallel Universes"

3.1 The Core Idea

Definition: Bootstrapping

Bootstrapping is a resampling technique that creates many simulated datasets by randomly sampling *with replacement* from our original data.

Key insight: Our original dataset represents the "population" (as best we know it). By resampling from it, we create variations that mimic what we'd see in parallel universes.

3.2 The Ball-in-Bucket Analogy

Example: Understanding Sampling with Replacement

Setup: You have a bucket with 5 numbered balls: {1, 3, 5, 8, 9}

Goal: Create a new "parallel universe" dataset of size 5

Process (Sampling WITH Replacement):

1. Reach in, randomly grab ball #8. Record it. **Put it back.**
2. Reach in again. Grab ball #8 again (possible because we replaced it!). Record.
3. Grab ball #3. Record. Put back.
4. Grab ball #5. Record. Put back.
5. Grab ball #1. Record.

Result:

- Original: {1, 3, 5, 8, 9}
- Bootstrap sample: {8, 8, 3, 5, 1}

Notice:

- Ball #8 appears *twice*
- Ball #9 doesn't appear at all
- This is exactly what we want—random variation!

Why "With Replacement"?

If we sampled *without* replacement, we'd just get the original dataset back (in a different order). That wouldn't create any variation!

Sampling *with* replacement means each draw is independent, and we naturally get variations where some points appear multiple times and others don't appear at all.

3.3 The Full Bootstrap Procedure

Bootstrap Algorithm for Confidence Intervals

Input: Original dataset of size n

Procedure:

1. Set number of bootstrap samples: S (typically 1000-10000)
2. For $s = 1$ to S:
   (a) Create bootstrap sample: randomly select n points from original data *with replacement*
   (b) Fit regression model to this bootstrap sample
   (c) Record coefficients: $\hat{\beta}_0^{(s)}, \hat{\beta}_1^{(s)}, \ldots$
3. Now you have S values of each coefficient
4. Analyze the distribution of these values

Output: Distribution of $\hat{\beta}$ values from which we can compute confidence intervals

3.4 Building Confidence Intervals from Bootstrap

Once we have S bootstrap estimates of $\hat{\beta}_1$, we can build a confidence interval:

Definition: Percentile Method for Confidence Intervals

95% Confidence Interval:

1. Sort all S bootstrap $\hat{\beta}$ values from smallest to largest
2. Find the 2.5th percentile (lower bound)
3. Find the 97.5th percentile (upper bound)
4. The interval between these is your 95% CI

In Python:

```
lower = np.percentile(bootstrap_betas, 2.5)
upper = np.percentile(bootstrap_betas, 97.5)
confidence_interval = [lower, upper]
```

Example: Computing Bootstrap CI

You ran 1000 bootstrap samples and got 1000 values of $\hat{\beta}_1$.

After sorting: [11.50, 12.26, 12.81, ..., 15.21]

- 2.5th percentile (25th value): 12.80
- 97.5th percentile (975th value): 13.71

95% CI: [12.80, 13.71]

Interpretation: We are 95% confident that the true β_1 lies between 12.80 and 13.71.

3.5 Standard Error

Definition: Standard Error

The **standard error** (SE) of $\hat{\beta}$ is simply the **standard deviation** of the bootstrap distribution.

$$SE_{\hat{\beta}} = \text{std}(\hat{\beta}^{(1)}, \hat{\beta}^{(2)}, \ldots, \hat{\beta}^{(S)})$$

If we assume the distribution is approximately normal, we can approximate:

$$95\%\ \text{CI} \approx [\bar{\beta} - 2 \cdot SE, \quad \bar{\beta} + 2 \cdot SE]$$

where $\bar{\beta}$ is the mean of bootstrap estimates.

Standard Deviation vs. Standard Error

Standard Deviation (SD): Measures spread of *data points* around their mean

- How spread out are the y values in our dataset?

Standard Error (SE): Measures spread of an *estimate* across hypothetical samples

- How much would $\hat{\beta}$ vary if we re-did the study many times?

In bootstrapping, SE is calculated as the standard deviation of the bootstrap $\hat{\beta}$ values.

4 Evaluating Predictor Significance

Now that we can quantify uncertainty in $\hat{\beta}$, we can ask more sophisticated questions about which predictors matter.

4.1 The Naive Approach: Largest Coefficient

Example: Advertising Data - Three Predictors

Suppose we have three predictors: TV, Radio, Newspaper.
After bootstrapping, we get:

Predictor	Mean $\hat{\beta}$	Std Dev (SE)
Newspaper	0.10	0.10
TV	0.05	0.005
Radio	-0.05	0.10

Naive question: Which predictor is most important?
Naive answer: Newspaper! It has the largest $|\hat{\beta}|$.
But wait... Look at the uncertainty! Newspaper's true β could be anywhere from -0.10 to +0.30. TV's is tightly concentrated around 0.05.

4.2 The t-test: Signal-to-Noise Ratio

We need a metric that considers **both** the coefficient value (signal) and its uncertainty (noise):

Definition: The t-test Statistic ($\hat{t}$)

The **t-test statistic** measures how many standard errors the coefficient is from zero:

$$\hat{t} = \frac{\text{Mean}(\hat{\beta})}{\text{SE}(\hat{\beta})} = \frac{\bar{\beta}}{\sigma_{\hat{\beta}}}$$

Interpretation:

- Large $|\hat{t}|$: Coefficient is far from zero relative to uncertainty → Strong signal
- Small $|\hat{t}|$: Coefficient is close to zero relative to uncertainty → Weak/uncertain signal

Note: Professor Protopapas uses $\hat{t}$ ("t-hat") because the classical t-statistic includes $\sqrt{n}$, which we omit for simplicity since n is fixed.

Example: Revisiting Advertising Data with t-test

Using the data from before:

Predictor	**Mean** $\hat{\beta}$	**SE**	$\hat{t} = \frac{\bar{\beta}}{SE}$
Newspaper	0.10	0.10	**1.0**
TV	0.05	0.005	**10.0**
Radio	-0.05	0.10	**-0.5**

New ranking by $|\hat{t}|$:

1. TV: $|\hat{t}| = 10$ (most important!)
2. Newspaper: $|\hat{t}| = 1$
3. Radio: $|\hat{t}| = 0.5$

Even though TV has the *smallest* coefficient, it's the *most reliably* non-zero!

Feature Importance Rankings Change!

The California Housing Price dataset example:

By coefficient magnitude $|\bar{\beta}|$:

1. Average Bedrooms
2. Median Income
3. Average Rooms

By t-statistic $|\hat{t}|$:

1. **Median Income**
2. **House Age**
3. Latitude

The coefficient-based ranking can be misleading! Average Bedrooms has a large coefficient but high uncertainty, so it drops in the t-statistic ranking.

5 Statistical Significance: The p-value

5.1 The Key Question

We've found that Median Income has the highest $\hat{t}$ score. But there's still a nagging question:

"What if ALL my predictors are junk, and Median Income is just the 'least bad'?"

We need to test whether the observed effect is **real** or just **random chance**.

5.2 Hypothesis Testing Framework

Definition: Hypothesis Testing

Null Hypothesis (H_0): The predictor has *no* effect on the outcome.

- Mathematically: $\beta = 0$
- Any non-zero $\hat{\beta}$ we observed was pure luck/noise

Alternative Hypothesis (H_1): The predictor *does* have an effect.

- Mathematically: $\beta \neq 0$

Strategy: Assume H_0 is true. Calculate how "surprising" our observed $\hat{t}$ would be under this assumption.

5.3 The p-value Concept

Definition: p-value

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

$$p\text{-value} = P(|t_{\text{random}}| \geq |t^*_{\text{observed}}| \mid H_0 \text{ is true})$$

In plain English: If there were truly no relationship, how often would random data produce a $\hat{t}$ value this large (or larger)?

5.4 How to Calculate p-value

Example: The p-value Calculation Process

Step 1: Generate random data (no relationship between x and y)
Step 2: Fit a model and calculate $\hat{t}$
Step 3: Repeat many times to build a distribution of "random $\hat{t}$ values"
Step 4: See where your actual $\hat{t}$ falls in this distribution
Shortcut: This distribution is the well-known **Student's t-distribution**. We don't need to simulate—we can compute directly!
Visualization: Plot the t-distribution. Your p-value is the area in the "tails" beyond your observed

$\hat{t}$ value (both positive and negative tails, since we consider absolute values).

5.5 Interpreting p-values

p-value Interpretation Guide

Large p-value (e.g., $p = 0.50$):

- "If there were no real effect, there's a 50% chance we'd see this result by luck."
- This is very plausible under H_0.
- **Conclusion**: Cannot reject H_0. No evidence the predictor matters.

Small p-value (e.g., $p = 0.01$):

- "If there were no real effect, there's only a 1% chance we'd see this result by luck."
- This is very unlikely under H_0.
- **Conclusion**: Reject H_0. The predictor is **statistically significant**.

Convention: We use $\alpha = 0.05$ as the threshold.

- $p < 0.05$: Reject H_0 (significant)
- $p \geq 0.05$: Cannot reject H_0 (not significant)

Common p-value Misconceptions

WRONG: "$p = 0.03$ means there's a 3% probability that H_0 is true."

CORRECT: "$p = 0.03$ means *if* H_0 were true, we'd see results this extreme only 3% of the time."

The p-value is about the *data*, not the hypothesis!

6 Prediction Intervals vs. Confidence Intervals

Now we shift from uncertainty in *coefficients* to uncertainty in *predictions*.

6.1 The Spaghetti Plot of Predictions

Example: Visualizing Prediction Uncertainty

From our S bootstrap samples, we have S different models:

$$\hat{f}^{(1)}(x), \hat{f}^{(2)}(x), \ldots, \hat{f}^{(S)}(x)$$

Plotting all S regression lines gives us a "spaghetti plot" for predictions.

For any specific x value (e.g., TV budget = \$200K):

- We get S different predicted values
- We can compute a histogram of these predictions
- We can compute the 2.5th and 97.5th percentiles $\rightarrow$ **Confidence Interval for** $f(x)$

6.2 The Key Distinction

Very Important: Confidence Interval vs. Prediction Interval

Confidence Interval (CI): Uncertainty in the *mean response* $f(x)$

- "Where does the *average* sales fall for TV budget = \$200K?"
- Only accounts for uncertainty in $\hat{\beta}$

Prediction Interval (PI): Uncertainty in a *new individual observation* y

- "What sales will a *specific new store* get with TV budget = \$200K?"
- Accounts for uncertainty in $\hat{\beta}$ **AND** the irreducible error ϵ

Key fact: **Prediction Interval is ALWAYS wider than Confidence Interval!**

Why? Because an individual y includes noise ϵ on top of the mean:

$$y = f(x) + \epsilon$$

Example: CI vs. PI Example

For TV Budget = \$200,000:

95% Confidence Interval for $f(x)$: [\$16.5M, \$17.5M]

- "The *average* sales for stores with this budget is 95% likely to be in this range."

95% Prediction Interval for y: [\$14.0M, \$20.0M]

- "A *specific new* store with this budget is 95% likely to have sales in this range."
- Much wider because we added uncertainty from ϵ!

6.3 The Funnel Shape

Both CI and PI have a characteristic "funnel" or "hourglass" shape:

- **Narrowest at** $\bar{x}$: Near the center of our data, we have the most information
- **Widens at extremes**: Far from the center, small errors in slope get amplified

This is because the regression line "pivots" around the data center $(\bar{x}, \bar{y})$.

7 Model Comparison Summary: When to Use What

This lecture began with comparing our three main model types:

Aspect	Linear Reg.	Polynomial Reg.	kNN
Type	Parametric	Parametric	Non-parametric
Interpretability	High	Moderate	Low
Coefficient meaning	Clear	Complex	None
Computational cost	Low	Moderate	High

Table 1: *Model comparison summary*

Key points:

- **Linear regression**: Fast (closed-form solution), interpretable

- **Polynomial regression**: Flexible but design matrix grows quickly
- **kNN**: Computationally expensive (must compute distances to all training points for every prediction)

8 Quick Reference Summary

Lecture 08 Quick Reference Card

1. Bootstrapping

- Create S "parallel universe" datasets by sampling *with replacement*
- Fit model to each $\rightarrow$ get distribution of $\hat{\beta}$
- **Confidence Interval**: Use 2.5th and 97.5th percentiles
- **Standard Error**: Standard deviation of bootstrap $\hat{\beta}$ values

2. Feature Importance

- **Naive**: Rank by $|\hat{\beta}|$ (ignores uncertainty!)
- **Better**: Rank by $|\hat{t}| = |\bar{\beta}|/SE$ (signal-to-noise ratio)

3. Statistical Significance

- **Null hypothesis** H_0: $\beta = 0$ (no effect)
- **p-value**: Probability of seeing our result if H_0 is true
- $p < 0.05 \rightarrow$ Reject $H_0 \rightarrow$ Significant!

4. CI vs. PI

- **Confidence Interval (CI)**: Uncertainty in mean $f(x)$
- **Prediction Interval (PI)**: Uncertainty in individual $y = f(x) + \epsilon$
- **PI is always wider** (includes ϵ variance)

9 Common Questions and Answers

Q: How many bootstrap samples (S) should I use?

A: Typically 1,000-10,000. More is better but has diminishing returns. For rough estimates, 1,000 is fine. For precise confidence intervals, use 10,000+.

Q: How does bootstrapping relate to overfitting?

A: Great question! Bootstrapping helps us understand coefficient uncertainty, but doesn't prevent overfitting. You should still use regularization (Ridge/Lasso) and cross-validation. In fact, you can bootstrap a Ridge regression model—apply regularization within each bootstrap iteration.

Q: When should I use bootstrap CIs vs. analytical formulas?

A: Analytical formulas (which assume normality) are faster but make assumptions. Bootstrap is more general—it works even when data isn't normal. Professor Protopapas prefers bootstrap because it's "assumption-free." In practice, both give similar results for large samples.

Q: Why does the confidence/prediction interval have a funnel shape?

A: The regression line is best constrained at the data center $(\bar{x}, \bar{y})$. Away from the center, small errors in the slope get amplified. Think of it like a see-saw pivoting at the center—small tilts at the pivot become large movements at the ends.

Q: Is $p < 0.05$ a universal rule?

A: No! It's a convention that works in many contexts, but:

- Some fields (particle physics) use much stricter thresholds
- Multiple testing requires adjustments (Bonferroni correction)
- Effect size matters too—a tiny effect can be "significant" with enough data

10 Looking Ahead

This lecture introduced key concepts that will be developed further:

- Kevin Rader will cover the **probabilistic foundations** of these ideas
- Formal hypothesis testing with assumptions about the error distribution
- Connection to the t-distribution and degrees of freedom
- These concepts extend to classification (logistic regression) and beyond

The key takeaway: Always quantify and communicate uncertainty. A point estimate without a confidence interval is incomplete!

Lecture 09: Probability and Maximum Likelihood Estimation

CS109A: Introduction to Data Science

Harvard University

■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 09
■ **Instructor:**	Pavlos Protopapas, Kevin Rader, Chris Gumb
■ **Objective:**	Understanding the probabilistic foundations of linear regression, connecting OLS to MLE, and comparing formula-based inference with bootstrapping

Contents

1 Introduction and Motivation

Lecture Overview

This lecture bridges the gap between machine learning and statistics. We'll discover that our "loss function" approach to linear regression has deep connections to probability theory—and this connection will help us understand when to trust our model's estimates.

Key Topics:

- **Probability Foundations**: Random variables, PMF, PDF
- **Key Distributions**: Normal (Gaussian) and Binomial
- **The Central Limit Theorem**: Why normal distributions are everywhere
- **Likelihood and MLE**: A new perspective on model fitting
- **The Big Connection**: OLS = MLE under normality assumptions
- **Formula-Based vs. Bootstrap Inference**: When each method shines

1.1 Today's Question: How Much is a House Worth?

Professor Rader motivates probability theory with a concrete question:

"What determines the selling price of a home in Cambridge/Somerville?"

Using data from Redfin.com (592 home sales), we'll explore:

- Response variable (Y): Selling price (in thousands of dollars)
- Predictors (X): Type (condo, single-family, etc.), bedrooms, bathrooms, square footage, lot size, year built, distance to Harvard Square T-stop

This real-world example will illustrate why probability theory matters for data science.

2 Exploratory Data Analysis: The Housing Dataset

2.1 Data Cleaning

Before modeling, we need to address data quality issues:

Data Preprocessing Steps

1. **Missing values in lot size**: Mostly condos/townhouses that don't own land
   - Solution: Impute with 0 (reasonable assumption)
2. **Missing values in HOA fees**: Mostly single-family homes without HOA
   - Solution: Impute with 0 (no HOA = no fees)
3. **Price scale**: Convert from dollars to thousands of dollars
   - Makes coefficients easier to interpret
4. **Zip code type**: Convert from numeric to categorical

- Zip codes don't have meaningful numerical order!

2.2 Key Observations from EDA

2.2.1 Heteroscedasticity

Definition: Heteroscedasticity

Heteroscedasticity occurs when the variance of residuals is not constant across all values of the predictor.

In this dataset: Smaller homes have less price variability. Larger homes have much more price variability.

This violates the assumption of constant variance in linear regression!

Example: Visualizing Heteroscedasticity

When plotting price vs. square footage:

- Small homes (1000 sqft): Prices cluster tightly, perhaps \$400K-\$600K
- Large homes (3000 sqft): Prices spread widely, perhaps \$800K-\$2M+

The "funnel" or "cone" shape is a classic sign of heteroscedasticity.

2.2.2 Collinearity Strikes Again

When fitting a multiple regression model, an interesting result emerges:

```
                coef    std err          t      P>|t|
Intercept  -1949.0670   745.203     -2.615      0.009
sqft           0.6411     0.044     14.720      0.000
beds         -89.9345    23.532     -3.822      0.000  <-- Negative!
baths        198.4646    31.332      6.334      0.000
```

The Negative Bedroom Coefficient

Why is the coefficient for "beds" negative?

This seems counterintuitive—shouldn't more bedrooms increase price?

Interpretation: "Holding *square footage constant*, each additional bedroom is associated with a \$89,934 *decrease* in price."

The insight: If you're cramming another bedroom into the *same* total square footage, you're creating smaller, more cramped rooms. Homes with lots of tiny bedrooms sell for less than homes with fewer, larger rooms (same total sqft).

This is collinearity at work—"beds" and "sqft" are highly correlated, so interpreting one while holding the other constant creates unusual but meaningful interpretations.

3 Review: Cross-Validation and Regularization

Before diving into probability, let's solidify some key concepts from previous lectures.

3.1 When to Use Cross-Validation

Cross-Validation is for Model Selection

Cross-validation can be used whenever you need to **choose between models**:

- **A. Choosing k in k-NN**: Different k values = different models
- **B. Choosing λ in Ridge/Lasso**: Different λ values = different models
- **C. Choosing predictors**: Different feature sets = different models
- **D. Choosing model families**: k-NN vs. linear regression = different models

Answer: ALL OF THE ABOVE!

Whenever you have a choice between models, cross-validation helps you make that choice objectively.

3.2 When to Standardize Predictors

Standardization Guidance

Standardize predictors when you want them to be **treated equally**:

- **k-NN**: Without standardization, variables on larger scales (price in dollars) dominate distance calculations over variables on smaller scales (number of rooms)
- **Ridge/Lasso**: Regularization penalizes coefficient magnitude. If one variable is measured in inches vs. feet, its coefficient scale changes artificially

But not always! If you have categorical dummies alongside continuous variables, you might *not* want to standardize everything equally. Use judgment!

3.3 Reading Trajectory Plots

Trajectory plots show how coefficients change as regularization strength (λ) increases:

- **Lasso**: Coefficients can become exactly zero (feature selection!)
- **Ridge**: Coefficients approach but never reach zero

"Textbook" vs. Real-World Trajectory Plots

Textbook plots: Smooth curves where all coefficients steadily shrink toward zero. These assume predictors are **independent**—unrealistic!

Real-world plots: Messy curves that may:

- Cross zero (sign changes!)
- Temporarily *increase* before decreasing

These patterns indicate **collinearity**: As one variable gets penalized, a correlated variable "picks up" its predictive power.

4 Probability Fundamentals

Now we build the probabilistic foundation needed to understand MLE.

4.1 What is Probability?

Definition: Probability

Probability is the long-run relative frequency of an event occurring.
Range: 0 (never happens) to 1 (always happens)
Why we care in data science: Our data is a **random realization** from some underlying data-generating process. Probability gives us the language to reason about uncertainty.

4.2 Random Variables

Definition: Random Variable

A **random variable** assigns numeric values to outcomes of a random phenomenon.
Example: Define $X_1 = 1$ if a randomly sampled Harvard student uses a Mac, $X_1 = 0$ otherwise. We don't know X_1's value until we sample—it's "random."
We describe random variables by their **distribution**—the set of possible values and their probabilities.

4.3 Discrete vs. Continuous: PMF vs. PDF

Random variables come in two flavors:

Aspect	**Discrete**	**Continuous**
Values	Countable (0, 1, 2, ...)	Any real number in a range
Function	PMF (Probability Mass Function)	PDF (Probability Density Function)
$P(X = x)$	Has a specific probability	**Always equals 0!**
Probabilities	Direct from PMF	Area under PDF curve
Example	Number of bedrooms	House price

Table 1: *Discrete vs. Continuous random variables*

Continuous Variables: $P(X$

For continuous random variables, the probability of any *exact* value is zero.
$P(\text{house price} = \$1,250,000.00) = 0$
Instead, we calculate probabilities over **intervals**:
$P(\$1,200,000 < \text{price} < \$1,300,000) = \text{area under PDF from 1.2M to 1.3M}$

5 Key Probability Distributions

5.1 The Bernoulli Distribution

The simplest distribution—a single coin flip:

Definition: Bernoulli Distribution

$X \sim \text{Bernoulli}(p)$

- $X = 1$ (success) with probability p
- $X = 0$ (failure) with probability $1 - p$

PMF: $P(X = x) = p^x(1-p)^{1-x}$ for $x \in \{0, 1\}$

Example: $X = 1$ if a student uses Mac, 0 otherwise

5.2 The Binomial Distribution

Multiple independent Bernoulli trials:

Definition: Binomial Distribution

$X \sim \text{Binomial}(n, p)$

Count of successes in n independent trials, each with success probability p.

PMF: $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is "n choose k"—the number of ways to arrange k successes among n trials.

Mean: $\mu = np$

Standard Deviation: $\sigma = \sqrt{np(1-p)}$

Example: Binomial Example

20% of Harvard students are varsity athletes. In a random sample of 200 students:

Expected number of athletes: $np = 200 \times 0.2 = 40$

Probability of exactly 50 athletes:

$$P(X = 50) = \binom{200}{50} (0.2)^{50} (0.8)^{150}$$

This can be computed in Python: `stats.binom.pmf(50, 200, 0.2)`

5.3 The Normal (Gaussian) Distribution

The most important distribution in statistics:

Definition: Normal Distribution

$X \sim N(\mu, \sigma^2)$

A continuous, bell-shaped distribution parameterized by:

- μ (mu): The mean (center of the bell)
- σ^2 (sigma squared): The variance (spread of the bell)

PDF: $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Standard Normal: $Z \sim N(0, 1)$ (mean 0, variance 1)
Standardization: Any $X \sim N(\mu, \sigma^2)$ can be converted to standard normal:

$$Z = \frac{X - \mu}{\sigma}$$

Z tells you "how many standard deviations away from the mean."

5.4 The Central Limit Theorem (CLT)

Very Important: Why Normal Distributions Are Everywhere

Central Limit Theorem (CLT):
Regardless of the original population distribution, the **sample mean** $\bar{X}$ of n independent observations approaches a normal distribution as n gets large:

$$\bar{X} \sim N\left(\mu, \frac{\sigma^2}{n}\right)$$

Key implications:

- The mean of $\bar{X}$ equals the population mean μ
- The variance of $\bar{X}$ **shrinks** as n increases (by factor of $1/n$)
- More data $\rightarrow$ more precise estimates!

Example: Why Human Height is Normally Distributed

Adult height results from the accumulation of many small factors:

- Genetics (many genes, each with small effect)
- Nutrition during childhood
- Sleep quality
- Exercise patterns
- And countless other factors...

By CLT, the "sum" of many small independent factors tends toward a normal distribution. This is why height histograms look bell-shaped!

6 From Probability to Inference: The Likelihood Function

6.1 Probability vs. Inference: Two Directions

The Two Directions of Statistical Reasoning

Probability (Deduction): Model → Data

- *Question*: "Given a fair coin ($p = 0.5$), what's the probability of getting 8 heads in 10 flips?"
- We know the model, we predict the data

Inference (Induction): Data → Model

- *Question*: "I flipped a coin 10 times and got 8 heads. Is this coin fair ($p = 0.5$) or biased?"
- We have data, we infer the model

Data science is mostly inference! We observe data and try to figure out what process generated it.

6.2 The Likelihood Function

Definition: Likelihood Function

The **likelihood function** is mathematically the same as the PMF/PDF, but with a different perspective:

- **PMF/PDF** $f(x|\theta)$: Fix parameters θ, vary data x
  - "Given this model, how probable is this data?"
- **Likelihood** $L(\theta|x)$: Fix data x, vary parameters θ
  - "Given this data, how plausible is this model?"

For independent observations:

$$L(\theta|x_1, \ldots, x_n) = \prod_{i=1}^{n} f(x_i|\theta)$$

The likelihood is a **product** of individual likelihoods (under independence).

6.3 Log-Likelihood: Making Life Easier

Products are mathematically inconvenient. Taking logs converts products to sums:

Definition: Log-Likelihood

The **log-likelihood** function:

$$\ell(\theta|x_1, \ldots, x_n) = \log L(\theta) = \sum_{i=1}^{n} \log f(x_i|\theta)$$

Why use log-likelihood?

1. Products become sums (much easier to work with!)
2. Since log is monotonically increasing, maximizing L is equivalent to maximizing ℓ

3. Numerical stability (avoids very small numbers from many multiplications)

6.4 Maximum Likelihood Estimation (MLE)

Definition: Maximum Likelihood Estimation

MLE finds the parameter value(s) that maximize the likelihood function:

$$\hat{\theta}_{MLE} = \arg\max_{\theta} L(\theta|\text{data}) = \arg\max_{\theta} \ell(\theta|\text{data})$$

Intuition: "Find the parameter that makes the observed data most probable."

Example: MLE for Normal Distribution

You observe three values: 3, 5, and 10. Assume they come from $N(\mu, 4)$ (variance = 4, unknown mean).

What's the MLE for μ?

Intuitively: The sample mean! $\hat{\mu}_{MLE} = \bar{x} = \frac{3+5+10}{3} = 6$

Why? The likelihood function (plotted against μ) is maximized when $\mu = \bar{x}$.

This isn't a coincidence—for normal distributions, the MLE of the mean is always the sample mean.

6.5 How to Find the MLE

Two approaches:

1. **Analytical (calculus)**: Take derivative of $\ell(\theta)$ with respect to θ, set equal to zero, solve
   - Works when closed-form solution exists
   - Example: Linear regression
2. **Numerical (optimization)**: Use algorithms like gradient descent to minimize **negative** log-likelihood
   - Works when no closed-form solution
   - Example: Logistic regression (coming soon!)

7 The Big Connection: OLS = MLE

This is the central insight of the lecture!

7.1 Setting Up the Probabilistic Model

Recall our linear regression model:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

Key assumption: The residuals ϵ_i are normally distributed:

$$\epsilon_i \sim N(0, \sigma^2)$$

This implies:

$$Y_i|X_i \sim N(\beta_0 + \beta_1 X_i, \sigma^2)$$

Each Y value, given its X, comes from a normal distribution centered at the regression line.

7.2 Building the Likelihood

For n independent observations:

$$L(\beta_0, \beta_1, \sigma^2|\text{data}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(Y_i - \beta_0 - \beta_1 X_i)^2}{2\sigma^2}\right)$$

7.3 The Log-Likelihood

Taking logs:

$$\ell(\beta_0, \beta_1, \sigma^2) = \underbrace{-\frac{n}{2}\log(2\pi\sigma^2)}_{\text{doesn't depend on } \beta} \underbrace{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i)^2}_{\text{depends on } \beta}$$

7.4 Maximizing the Log-Likelihood

To find $\hat{\beta}_0, \hat{\beta}_1$ that maximize ℓ:

- The first term doesn't involve β, so it's irrelevant for optimization
- The second term has a negative sign, so maximizing ℓ means **minimizing** the sum

Very Important: The Central Result: OLS

Maximizing the likelihood (MLE) with respect to β_0, β_1 is equivalent to minimizing:

$$\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i)^2$$

This is exactly the Sum of Squared Errors (SSE)!
Conclusion: If residuals are normally distributed, then:

Ordinary Least Squares (OLS) = Maximum Likelihood Estimation (MLE)

This provides the **probabilistic justification** for why we use MSE as our loss function!

8 Statistical Inference: Quantifying Uncertainty

Now that we have the probabilistic framework, we can quantify uncertainty in our estimates.

8.1 Point Estimates Are Not Enough

We computed $\hat{\beta}_1 = 0.5898$ for the square footage coefficient. But:

- This is based on one sample of 592 homes
- A different sample would give a different $\hat{\beta}_1$
- How confident should we be in this specific value?

We need to quantify this uncertainty.

8.2 Two Approaches to Inference

1. **Bootstrap** (from previous lecture): Resample data, compute estimates, look at distribution
2. **Formula-based** (this lecture): Use mathematical formulas derived from probability theory

8.3 Formula-Based Confidence Intervals

Under the linear regression assumptions, we have closed-form formulas for the **standard error** of $\hat{\beta}_1$:

Definition: Standard Error of the Slope

$$\hat{SE}(\hat{\beta}_1) = \sqrt{\frac{\hat{\sigma}^2}{\sum_{i=1}^{n}(X_i - \bar{X})^2}}$$

where $\hat{\sigma}^2 = \frac{\sum(Y_i - \hat{Y}_i)^2}{n-p-1}$ (corrected MSE)

Interpretation:

- More data (n larger) $\rightarrow$ smaller SE (more precise!)
- More spread in X (larger $\sum(X_i - \bar{X})^2$) $\rightarrow$ smaller SE
- Better model fit (smaller $\hat{\sigma}^2$) $\rightarrow$ smaller SE

Definition: Confidence Interval Formula

$$95\% \text{ CI for } \beta_1 = \hat{\beta}_1 \pm t^* \cdot \hat{SE}(\hat{\beta}_1)$$

where $t^* \approx 2$ for 95% confidence (from the t-distribution).

Interpretation: If we repeated the study many times, approximately 95% of the intervals we construct would contain the true β_1.

8.4 Hypothesis Testing

Hypothesis Testing for Regression Coefficients

Hypotheses:

- H_0: $\beta_1 = 0$ (no association between X and Y)
- H_A: $\beta_1 \neq 0$ (there is an association)

Test statistic:

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

Interpretation: How many standard errors is our estimate from zero?

p-value: Probability of seeing a $|t|$ this large or larger if H_0 were true.

Decision: If p-value < 0.05, reject H_0. Conclude the association is statistically significant.

9 Bootstrap vs. Formula-Based Inference

Let's compare the two approaches on our housing data:

9.1 The Comparison

For the square footage coefficient ($\hat{\beta}_1 = 0.5898$):

Method	95% CI	Width
Bootstrap	[0.487, 0.705]	0.218
Formula (statsmodels)	[0.544, 0.636]	0.092

Table 2: *Confidence intervals: Bootstrap vs. Formula-based*

Why the Difference?

The formula-based CI is much **narrower**—it suggests we're more confident than we should be!
The problem: Formula-based inference assumes all linear regression assumptions hold, including **constant variance (homoscedasticity)**.
But we already identified **heteroscedasticity** in this data—larger homes have more variable prices!
When assumptions are violated, the formulas give **incorrect standard errors**, leading to **overly optimistic** (too narrow) confidence intervals.

9.2 When to Use Each Method

Very Important: Choosing Your Inference Method

Use formula-based inference when:

- All regression assumptions are reasonably met
- You want fast computation
- Results are easy to report (standard output in statsmodels)

Use bootstrap when:

- Assumptions may be violated (especially heteroscedasticity)
- You want robust inference without strong distributional assumptions
- The bootstrap captures the "true" variability in your data

Bottom line: Bootstrap is **safer**—it makes fewer assumptions and reflects the actual data distribution.

10 Using statsmodels for Inference

```
1 import statsmodels.api as sm
2 import statsmodels.formula.api as smf
```

```python

# Using formula interface (like R!)
model = smf.ols('price ~ sqft + beds + baths + type', data=df)
results = model.fit()

# Get full summary with standard errors, t-stats, p-values, CIs
print(results.summary())

# Extract specific values
print(f"Coefficient for sqft: {results.params['sqft']:.4f}")
print(f"Std Error: {results.bse['sqft']:.4f}")
print(f"p-value: {results.pvalues['sqft']:.4f}")
print(f"95% CI: {results.conf_int().loc['sqft'].values}")
```

Listing 1: Fitting regression with statsmodels

statsmodels vs. sklearn

sklearn: Great for prediction, cross-validation, pipelines

- Doesn't provide standard errors, p-values, or confidence intervals
- Focus on predictive performance

statsmodels: Great for inference and interpretation

- Provides full statistical output
- R-like formula interface for easy model specification
- Automatically creates dummy variables for categorical predictors

Use both! sklearn for prediction tasks, statsmodels for understanding relationships.

11 Quick Reference Summary

Lecture 09 Quick Reference Card

1. Key Distributions

- **Bernoulli**: Single binary outcome (p)
- **Binomial**: Count of successes in n trials (n, p)
- **Normal**: Bell-shaped continuous (μ, σ^2)

2. Central Limit Theorem

Sample means approach normal distribution as n increases:

$$\bar{X} \sim N\left(\mu, \frac{\sigma^2}{n}\right)$$

3. The Big Connection

Under assumption of normal residuals:

$$\text{OLS (minimize MSE)} \equiv \text{MLE}$$

This justifies using MSE as our loss function!

4. Inference Methods

- **Formula-based**: Fast, but assumes all assumptions hold
- **Bootstrap**: Robust, makes fewer assumptions
- When assumptions violated: **Bootstrap wins!**

5. Confidence Interval Formula

$$\hat{\beta}_1 \pm t^* \cdot \hat{SE}(\hat{\beta}_1)$$

where $t^* \approx 2$ for 95% CI

12 Common Questions and Answers

Q: Why do we care about the probabilistic interpretation of OLS?

A: It gives us:

1. Theoretical justification for using MSE
2. A framework for statistical inference (standard errors, p-values, CIs)
3. Understanding of what assumptions we're making

Q: If bootstrap is more robust, why ever use formulas?

A: Formulas are much faster (instant computation vs. 1000+ resamples). When assumptions hold, they give the same answer. For quick checks or when you're confident in assumptions, formulas are fine.

Q: How do I know if heteroscedasticity is a problem?

A: Plot residuals vs. fitted values (or vs. X). Look for "funnel" shapes or patterns. If you see non-constant spread, heteroscedasticity is present. Use bootstrap for inference!

Q: Why is the coefficient for "beds" negative?

A: Collinearity with square footage. "Holding sqft constant, more bedrooms means cramming smaller rooms into the same space"—which decreases value. Always interpret coefficients in the context of "holding other variables constant."

Q: What percentage of Harvard students use Macs?

A: Professor Rader's informal estimate: around 75-90%. This is used as a fun example for binomial distributions!

13 Looking Ahead

In the next lectures, we'll:

- Continue with statistical inference in more detail
- Move to **classification** problems (predicting categories, not numbers)
- Introduce **logistic regression**—where MLE doesn't have a closed-form solution

The probability foundation we built today will be essential for understanding logistic regression's likelihood function!

Lecture 10: Bayesian Inference and Bayes' Rule

CS109A: Introduction to Data Science

Harvard University

■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 10
■ **Instructor:**	Pavlos Protopapas, Kevin Rader, Chris Gumb
■ **Objective:**	Reviewing statistical inference, understanding the distinction between confidence and prediction intervals, learning Bayes' rule, and introducing Bayesian inference as an alternative paradigm to frequentist statistics

Contents

1 Introduction and Motivation

Lecture Overview

This lecture reviews statistical inference concepts and introduces a fundamentally different way of thinking about probability and parameters: **Bayesian inference**. The Bayesian approach treats parameters as random variables with probability distributions, rather than fixed unknown constants.

Key Topics:

- **Review of Inference**: Standard errors, confidence intervals, hypothesis testing
- **Bootstrap vs. Permutation**: When to use which resampling method
- **Confidence Interval vs. Prediction Interval**: A crucial distinction
- **Likelihood Review**: The foundation for Bayesian methods
- **Bayes' Rule**: Flipping conditional probabilities
- **Bayesian Inference**: Updating beliefs based on evidence
- **Frequentist vs. Bayesian**: Two worldviews of statistics

1.1 Continuing the Housing Example

We continue with the Cambridge/Somerville housing data from last lecture:

- **Simple regression**: Price $\sim$ Square footage
- **Estimated model**: $\hat{y} = 247.4 + 0.5898x$
- **Interpretation**: Each additional square foot is associated with roughly \$600 higher selling price

Correlation vs. Causation Reminder

"Be careful. Doesn't mean it's causal because there could be other confounders in that data set that we haven't controlled for."

The coefficient 0.5898 represents an *association*, not necessarily a causal effect. Other factors correlated with square footage (neighborhood quality, lot size, etc.) might be driving part of this relationship.

2 Review: Statistical Inference

2.1 Population vs. Sample

Definition: Two Models

Population Model (what we want):

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

- β_0, β_1 are the *true* parameters for all homes in this geographic region

- $\sigma^2 = \text{Var}(\epsilon_i)$ is another unknown parameter

Estimated Model (what we have):

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

- $\hat{\beta}_0, \hat{\beta}_1$ are estimates from our sample of ~500 homes
- These are one "realization" from the sampling distribution

2.2 Standard Errors: Understanding Uncertainty

The **standard error** (SE) quantifies how much our estimate would vary across different samples:

Definition: Standard Error of the Slope

$$\hat{SE}(\hat{\beta}_1) = \sqrt{\frac{\hat{\sigma}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}}$$

where $\hat{\sigma}^2$ is the estimated residual variance.

2.2.1 What Controls the Standard Error?

Looking at the formula, we can build intuition:

How to Reduce Standard Error

1. **Increase sample size (n):**
   - More observations → larger denominator
   - The sum $\sum(x_i - \bar{x})^2$ increases with n
   - **Most reliable way to reduce SE**
2. **Increase spread in X:**
   - Wider range of X values → larger $\sum(x_i - \bar{x})^2$
   - Hard to control in observational studies
   - In experiments, you can deliberately sample extreme X values
3. **Reduce residual variance ($\hat{\sigma}^2$):**
   - Better model → tighter fit around the line
   - Add relevant predictors, use transformations
   - Limited by irreducible error

2.2.2 What About Standardizing Predictors?

Professor Rader poses a "midterm type question": What happens to the SE formula if you standardize the predictor?

Example: The Standardization Trap

At first glance, standardizing X (mean 0, variance 1) might seem to reduce SE since the denominator becomes smaller.

But wait!

When you standardize X:

- The denominator $\sum(x_i - \bar{x})^2$ decreases
- BUT $\hat{\beta}_1$ itself changes (different units!)
- Interpretation changes from "per unit X" to "per standard deviation of X"

Net effect: The t-statistic remains the same!

$$t = \frac{\hat{\beta}_1}{\hat{SE}(\hat{\beta}_1)}$$

Standardizing doesn't magically make your predictor more "significant." The narrower CI is offset by a smaller $\hat{\beta}_1$.

2.3 Confidence Intervals: Formula-Based

Definition: Confidence Interval Formula

$$\text{CI for } \beta_1: \quad \hat{\beta}_1 \pm t^* \cdot \hat{SE}(\hat{\beta}_1)$$

where t^* is the critical value from the t-distribution (roughly 2 for 95% CI with large samples).

2.3.1 Why t-distribution Instead of Normal?

t-Distribution vs. Normal Distribution

Normal (Z) distribution: Used when σ is *known* (rare in practice).

t-distribution: Used when σ must be *estimated* from data.

- Estimating σ adds extra uncertainty
- t-distribution has "fatter tails" to account for this
- As sample size $\to \infty$, t-distribution $\to$ normal distribution
- With $n \geq 50$, the difference is minimal

2.4 Hypothesis Testing Review

Hypothesis Testing for β_1

Hypotheses:

- $H_0 : \beta_1 = 0$ (no association between X and Y)
- $H_A : \beta_1 \neq 0$ (there is an association)

Test Statistic:

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

This measures: "How many standard errors is our estimate from zero?"
p-value: Probability of observing a $|t|$ this extreme or more extreme, assuming H_0 is true.
Decision: If p-value < 0.05, reject H_0.

2.5 Two-Sided Tests and Absolute Values

Example: Why Absolute Values in p-value Calculation?

The p-value formula often looks like:

$$\text{p-value} = P(|T| \geq |t_{\text{observed}}|)$$

Why absolute values?
Because our alternative hypothesis is $\beta_1 \neq 0$ (two-sided). We consider evidence against H_0 whether the association is positive or negative.
If we observed $t = -4$:

- Area to the left of -4 (extreme negative)
- Plus area to the right of $+4$ (equally extreme positive)

3 Bootstrap vs. Permutation Tests

3.1 When to Use Each

Very Important: Bootstrap vs. Permutation: Critical Distinction

Bootstrap: For **confidence intervals**

- Resample *with replacement* from your data
- Preserves the relationships in your data
- Estimates the sampling distribution of your statistic

Permutation Test: For **hypothesis testing (p-values)**

- Shuffle the response variable Y, keep X fixed
- **Enforces the null hypothesis** (no relationship between X and Y)
- Builds a reference distribution *under* H_0

Why not bootstrap for hypothesis testing?
Bootstrap doesn't enforce H_0. Under certain conditions (violated assumptions, multicollinearity), it can lead to **inflated Type I error**—rejecting H_0 too often when it's actually true.

Example: Permutation Testing Procedure

To test $H_0 : \beta_1 = 0$:

1. **Shuffle**: Randomly permute the Y values (break the X-Y relationship)
2. **Refit**: Calculate $\hat{\beta}_1^*$ on the shuffled data
3. **Repeat**: Do this many times (e.g., 1000 permutations)
4. **Result**: A distribution of $\hat{\beta}_1^*$ values, centered at 0 (since H_0 is enforced)

The p-value is the proportion of permuted $|\hat{\beta}_1^*|$ values that exceed your observed $|\hat{\beta}_1|$.

3.2 Comparing Bootstrap and Formula-Based CIs

From the housing data:

Method	**95% CI for β_1**	**Width**
statsmodels (formula)	(0.544, 0.636)	0.092
Bootstrap	(0.487, 0.705)	0.218

Table 1: *Comparison of confidence intervals*

Why the Difference?

The bootstrap CI is **wider** because:

- The formula-based CI assumes constant variance (homoscedasticity)
- Our data shows clear heteroscedasticity (variance fanning out with larger homes)
- When assumptions are violated, formulas give **incorrectly narrow** CIs
- Bootstrap makes fewer assumptions and captures the true variability

Lesson: When in doubt, bootstrap is safer!

3.3 Linear Regression Assumptions (LINE)

The LINE Assumptions

1. **L**inearity: The relationship is linear
2. **I**ndependence: Observations are independent (most important!)
3. **N**ormality: Residuals are normally distributed (least important if $n > 50$)
4. **E**qual variance: Constant variance (homoscedasticity)

Bootstrap: Relaxes N and E, still requires L and I

Formula-based: Requires all four

4 Confidence Interval vs. Prediction Interval

This is one of the most commonly confused distinctions in regression!

4.1 Two Different Questions

Definition: CI vs. PI

Confidence Interval (CI): Uncertainty about the *mean* response

- Question: "What is the **average** selling price of *all* homes with 2860 sqft?"
- Only includes uncertainty in the regression line ($\hat{\beta}$'s)

Prediction Interval (PI): Uncertainty about a *single* new observation

- Question: "What will *this specific new home* with 2860 sqft sell for?"
- Includes uncertainty in the line **AND** the individual noise (σ^2)

4.2 Why PI is Always Wider

Two Sources of Uncertainty for Predictions

Source 1: Uncertainty in the regression line

- We estimated $\hat{\beta}_0$ and $\hat{\beta}_1$ from a sample
- Different samples would give different estimates
- This is what the CI captures

Source 2: Individual observation variability (irreducible error)

- Even if we knew the exact population line, individual homes vary around it
- This is σ^2 (the variance of ϵ)
- No matter how much data you collect, this never goes away!

Prediction Interval:

$$\text{PI} = \hat{y} \pm t^* \cdot \sqrt{(\text{SE of fit})^2 + \hat{\sigma}^2}$$

The extra $\hat{\sigma}^2$ term is why PI is always wider than CI.

Example: Game Time Question

Use this output to predict with 95% uncertainty the selling price of a home that is 2860 sqft:

```
Intercept: 247.4,  sqft coef: 0.5898,  SE(sqft): 0.023
```

Options:

A. $0.5898 \pm 2 \times 0.023$ (CI for slope)

B. $247.4 + 0.5898 \times 2860$ (point prediction only)

C. $247.4 + 0.5898(2860) \pm 2 \times 0.023$ (CI for mean response)

D. $247.4 + 0.5898(2860) \pm 2\sqrt{0.023^2 + \hat{\sigma}^2}$ (PI for new observation)

Answer: **D**!

Since we're predicting "a home"—a single new observation—we need the **Prediction Interval**, which includes both the model uncertainty and the residual variance $\hat{\sigma}^2$.

5 Likelihood Review

Before diving into Bayesian inference, let's solidify our understanding of likelihood.

5.1 Flipping the Perspective

Definition: Likelihood Function

PDF/PMF: $f(x|\theta)$ — Given parameters, what's the probability of the data?
Likelihood: $L(\theta|x)$ — Given data, how plausible is each parameter value?
Mathematically: They're the same function! The difference is conceptual:

- PDF: θ is fixed, x varies
- Likelihood: x is fixed (observed), θ varies

5.2 Why Products Become Sums

Log-Likelihood

For independent observations $x_1, \ldots, x_n$:

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

Taking the log:

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log f(x_i|\theta)$$

Why use log?

1. Products $\rightarrow$ sums (much easier calculus!)
2. Numerical stability (avoids underflow from multiplying many small numbers)
3. Same maximum: $\arg\max L(\theta) = \arg\max \ell(\theta)$

5.3 The OLS-MLE Connection (Review)

OLS

If we assume $\epsilon_i \sim N(0, \sigma^2)$, then maximizing the likelihood is equivalent to minimizing the sum of squared errors.
Why? The negative log-likelihood becomes:

$$-\ell(\beta_0, \beta_1) = \text{constant} + \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

Minimizing this with respect to β_0, β_1 is the same as minimizing SSE!

6 Bayes' Rule: Flipping Conditional Probabilities

6.1 The Basic Formula

Definition: Bayes' Rule

For events A and B:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Extended version (using Law of Total Probability for denominator):

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B|A) \cdot P(A) + P(B|A^c) \cdot P(A^c)}$$

6.2 Example: CS and STAT Concentrators

Example: Conditional Probability Practice

In a hypothetical CS109A class (among undergrads):

- 40% are STAT concentrators: $P(\text{STAT}) = 0.40$
- 60% are CS concentrators: $P(\text{CS}) = 0.60$
- 20% are both (joint/double): $P(\text{STAT} \cap \text{CS}) = 0.20$

Question 1: Among STAT concentrators, what fraction are also CS?

$$P(\text{CS}|\text{STAT}) = \frac{P(\text{STAT} \cap \text{CS})}{P(\text{STAT})} = \frac{0.20}{0.40} = 0.50$$

Question 2: Among CS concentrators, what fraction are also STAT?

$$P(\text{STAT}|\text{CS}) = \frac{P(\text{STAT} \cap \text{CS})}{P(\text{CS})} = \frac{0.20}{0.60} = 0.333$$

Key Insight: $P(\text{CS}|\text{STAT}) \neq P(\text{STAT}|\text{CS})$!

These are fundamentally different questions—don't confuse them.

6.3 Independence and Dependence

Are STAT and CS Independent?

Events are **independent** if $P(A \cap B) = P(A) \cdot P(B)$.

Check: $P(\text{STAT}) \cdot P(\text{CS}) = 0.40 \times 0.60 = 0.24$

But $P(\text{STAT} \cap \text{CS}) = 0.20 \neq 0.24$

Conclusion: They are **dependent**!

Meaning: Knowing someone's CS status gives you information about whether they're also STAT. The probability changes once you have additional information.

7 Bayes' Rule in Diagnostic Testing

This is a classic application that will connect directly to classification later.

Example: Pregnancy Test Example

A pregnancy test has:

- **Sensitivity**: $P(\text{Test} + | \text{Pregnant}) = 0.97$ (true positive rate)
- **Specificity**: $P(\text{Test} - | \text{Not Pregnant}) = 0.99$ (true negative rate)

Among people taking this test, about 30% are actually pregnant: $P(\text{Pregnant}) = 0.30$

Question: If someone tests positive, what's the probability they're actually pregnant?

We want: $P(\text{Pregnant}|\text{Test}+)$

We have: $P(\text{Test} + | \text{Pregnant})$

We need to **flip the conditional**—use Bayes' Rule!

7.1 Applying Bayes' Rule

$$
\begin{aligned}
P(\text{Preg}|T+) &= \frac{P(T+|\text{Preg}) \cdot P(\text{Preg})}{P(T+)} \\
&= \frac{P(T+|\text{Preg}) \cdot P(\text{Preg})}{P(T+|\text{Preg}) \cdot P(\text{Preg}) + P(T+|\text{Not Preg}) \cdot P(\text{Not Preg})} \\
&= \frac{0.97 \times 0.30}{0.97 \times 0.30 + 0.01 \times 0.70} \\
&= \frac{0.291}{0.291 + 0.007} = \frac{0.291}{0.298} \approx 0.977
\end{aligned}
$$

Belief Update: Prior to Posterior

- **Prior probability** of being pregnant: 30%
- **Evidence**: Positive test result
- **Posterior probability** of being pregnant: 97.7%

The positive test dramatically updated our belief from 30% to 97.7%!

This is the essence of Bayesian inference: Using evidence to update our beliefs.

8 Bayesian Inference: A New Paradigm

8.1 From Events to Parameters

Now we apply this same logic to statistical parameters:

Definition: Bayesian Inference Formula

$$f(\theta|X) = \frac{f(X|\theta) \cdot f(\theta)}{f(X)}$$

Components:

- $f(\theta|X)$ — **Posterior Distribution**: Our updated belief about θ after seeing data
- $f(X|\theta)$ — **Likelihood**: How probable is our data given θ?
- $f(\theta)$ — **Prior Distribution**: Our initial belief about θ before seeing data
- $f(X)$ — **Evidence/Marginal Likelihood**: A normalizing constant

Simplified:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

8.2 The Bayesian Philosophy

Very Important: Frequentist vs. Bayesian Worldviews

8.3 Example: Three Coins

Example: Discrete Bayesian Inference

You have three coins in your pocket:

- Coin A: $P(\text{Heads}) = 0.1$ (biased toward tails)
- Coin B: $P(\text{Heads}) = 0.5$ (fair)
- Coin C: $P(\text{Heads}) = 0.9$ (biased toward heads)

You randomly select one coin (equal probability) and flip it 4 times.

Data observed: 3 Heads, 1 Tail

Question: Which coin did you probably pick?

Step 1: Prior

Before seeing any flips, each coin is equally likely:

$$P(p = 0.1) = P(p = 0.5) = P(p = 0.9) = \frac{1}{3}$$

Step 2: Likelihood

Calculate probability of "3H, 1T" for each coin using binomial distribution:

$$L(p = 0.1) = \binom{4}{3}(0.1)^3(0.9)^1 = 4 \times 0.001 \times 0.9 = 0.0036$$

$$L(p = 0.5) = \binom{4}{3}(0.5)^3(0.5)^1 = 4 \times 0.125 \times 0.5 = 0.2500$$

$$L(p = 0.9) = \binom{4}{3}(0.9)^3(0.1)^1 = 4 \times 0.729 \times 0.1 = 0.2916$$

Step 3: Posterior (unnormalized)

$$P(p = 0.1|\text{data}) \propto 0.0036 \times \frac{1}{3} = 0.0012$$
$$P(p = 0.5|\text{data}) \propto 0.2500 \times \frac{1}{3} = 0.0833$$
$$P(p = 0.9|\text{data}) \propto 0.2916 \times \frac{1}{3} = 0.0972$$

Step 4: Normalize (sum = 0.0012 + 0.0833 + 0.0972 = 0.1817)

$$P(p = 0.1|\text{data}) = \frac{0.0012}{0.1817} \approx 0.007 \quad (0.7\%)$$
$$P(p = 0.5|\text{data}) = \frac{0.0833}{0.1817} \approx 0.458 \quad (45.8\%)$$
$$P(p = 0.9|\text{data}) = \frac{0.0972}{0.1817} \approx 0.535 \quad (53.5\%)$$

Conclusion: Our belief shifted from (33.3%, 33.3%, 33.3%) to (0.7%, 45.8%, 53.5%). We now believe we most likely picked the biased-toward-heads coin!

9 Game Time: Bootstrap Sample Size

Example: Class Question

What happens to the bootstrap distribution when B (number of bootstrap samples) increases?
Options:

A. The distribution becomes more normal

B. The variance decreases

C. The confidence intervals become narrower

D. The distribution gets smoother

Correct Answer: **D** (and arguably A)

What increasing B does:

- Gets a more *precise estimate* of the true sampling distribution
- Makes the histogram smoother (less jagged)
- Does NOT change the underlying variability of your estimate!

What increasing B does NOT do:

- Narrow confidence intervals
- Reduce the variance of $\hat{\beta}$

To actually narrow CIs, you need to increase n (original sample size), not B!

10 Quick Reference Summary

Lecture 10 Quick Reference Card

1. CI vs. PI

- **CI**: Uncertainty in mean response (where is the line?)
- **PI**: Uncertainty in single observation (includes $\hat{\sigma}^2$)
- PI is **always wider**!

2. Bootstrap vs. Permutation

- **Bootstrap**: Confidence intervals (resample with replacement)
- **Permutation**: Hypothesis tests (shuffle Y, enforces H_0)

3. Bayes' Rule

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Allows "flipping" conditional probabilities!

4. Bayesian Inference

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

$$f(\theta|X) \propto f(X|\theta) \cdot f(\theta)$$

5. Frequentist vs. Bayesian

- **Frequentist**: θ is fixed constant, probability = long-run frequency
- **Bayesian**: θ has a distribution, probability = belief

11 Common Questions and Answers

Q: The prior seems subjective. Isn't that a problem?

A: It can be, but:

1. Use "informative priors" based on previous research or domain knowledge
2. Use "uninformative/flat priors" when you have no prior knowledge
3. Most importantly: **With enough data, the prior gets overwhelmed**. The posterior converges to the same answer regardless of (reasonable) prior choices.

Q: When should I use Bayesian vs. Frequentist methods?

A: Both have their place:

- **Frequentist**: Simpler, faster computation, standard in clinical trials

- **Bayesian**: Can incorporate prior knowledge, gives full posterior distribution, better for small samples

Modern data science often uses both and compares results.

Q: Why does increasing bootstrap samples B not narrow the CI?

A: B controls how accurately you *estimate* the sampling distribution. But the *width* of the sampling distribution depends on your original sample size n. More bootstrap samples give you a smoother histogram of the same width.

12 The Monty Hall Problem (Bonus)

Professor Rader mentions this classic probability puzzle:

Example: The Monty Hall Problem

Setup:

- 3 doors: 1 car (prize), 2 goats (losers)
- You pick a door (say, Door 1)
- Host (who knows where the car is) opens another door revealing a goat
- You're offered: "Do you want to switch to the remaining door?"

Counterintuitive Result:

- Stay: Win probability = 1/3
- Switch: Win probability = 2/3

Intuition (100 doors version):

- Pick 1 door out of 100. Probability you picked the car = 1/100.
- Host reveals 98 goats, leaving 2 doors.
- If you switch, you win unless you initially picked the car (probability 99/100)!

This demonstrates how conditional probability can be counterintuitive!

13 Looking Ahead

In the next lecture, we'll:

- Work through the three-coins example in more detail
- Explore continuous priors and posteriors
- See how Bayesian inference connects to regression
- Start applying these ideas in Python

The Bayesian framework will also be crucial when we move to classification, where we'll care about $P(\text{Class}|\text{Features})$—a natural application of Bayes' Rule!

Lecture 11: Bayesian Modeling

CS109A: Introduction to Data Science

Harvard University

Course:	CS109A: Introduction to Data Science
Lecture:	Lecture 11
Instructors:	Pavlos Protopapas, Kevin Rader, Chris Gumb
Topics:	Bayesian inference, Bayes' Rule, Prior and Posterior distributions, Normal-Normal model, Conjugate priors, Connection to Ridge and Lasso regression

Contents

Lecture Overview

This lecture introduces **Bayesian modeling**, a fundamentally different approach to statistical inference that treats parameters as random variables with probability distributions. Instead of finding a single "best estimate," Bayesian methods provide a complete probability distribution representing our uncertainty about parameters.

Key Learning Objectives:

- Understand the difference between Frequentist and Bayesian perspectives
- Master Bayes' Rule and its application to inference
- Learn about prior distributions and their role in modeling
- Understand the Normal-Normal conjugate model
- Discover the deep connection between Bayesian inference and regularization (Ridge/Lasso)

1.1 Two Paradigms of Statistics

Statistics has two major schools of thought about how to approach inference: the **Frequentist** approach and the **Bayesian** approach. Both are valid and useful, but they have fundamentally different philosophies.

Definition: Frequentist Statistics

In the frequentist approach:

- **Parameters are fixed but unknown constants.** There exists one "true" value of θ that we're trying to estimate.
- **Data is random.** Each time we collect data, we get a random sample from some population.
- **Probability means long-run frequency.** When we say "95% confidence," we mean that if we repeated the experiment infinitely many times, 95% of our intervals would contain the true parameter.
- **Inference is about the sampling distribution.** We reason about what would happen across many hypothetical repetitions of our experiment.

Definition: Bayesian Statistics

In the Bayesian approach:

- **Parameters are random variables.** We express our uncertainty about θ using probability distributions.
- **Data is fixed (once observed).** After we collect our data, it's no longer random—it's what we actually observed.
- **Probability means degree of belief.** When we say "95% probability," we mean we're 95% confident that θ is in a certain range.
- **Inference is about updating beliefs.** We start with prior beliefs and update them based on observed data.

Example: Treasure Hunt Analogy

Imagine you're searching for buried treasure.

Frequentist Approach:

- There is ONE exact location where the treasure is buried (fixed, unknown parameter)
- You collect clues (data) and estimate the location
- Your estimate is a single point: "The treasure is at coordinates (100, 200)"

Bayesian Approach:

- You start with a "probability map" showing where you think the treasure might be
- As you gather clues (data), you **update your map**
- Your result is the entire updated map: "60% chance it's in region A, 30% in region B, 10% in region C"

1.2 Why Learn Bayesian Methods?

Bayesian modeling offers several advantages:

1. **Intuitive interpretation:** "There's a 95% probability that θ is between 2 and 5" is more natural than the frequentist confidence interval interpretation.
2. **Incorporating prior knowledge:** You can formally include expert opinion, historical data, or physical constraints into your model.

3. **Flexibility:** Bayesian methods can handle complex hierarchical models where data is measured at multiple levels.
4. **Sequential updating:** As new data arrives, you can update your beliefs continuously without starting from scratch.
5. **Connection to regularization:** Ridge and Lasso regression have elegant Bayesian interpretations.
6. **Meta-analysis:** Combining results from multiple studies is natural in the Bayesian framework.

Important Note

When to Choose Each Approach

Frequentist methods are often preferred when:

- You need quick, computationally efficient solutions
- Classical approaches are well-established in your field
- You want to avoid specifying prior beliefs

Bayesian methods are often preferred when:

- You have relevant prior information to incorporate
- You need probabilistic statements about parameters
- Your model is hierarchical or complex
- Data arrives sequentially

In practice, **both methods often give similar results** when the prior is uninformative and the sample size is large.

2 Bayes' Rule: The Foundation

Everything in Bayesian inference flows from a single formula: **Bayes' Rule**.

2.1 The Formula

Definition: Bayes' Rule

For parameters θ and data X:

$$P(\theta|X) = \frac{P(X|\theta) \cdot P(\theta)}{P(X)} \tag{1}$$

Or in terms of probability density functions (PDFs):

$$f(\theta|X) = \frac{f(X|\theta) \cdot f(\theta)}{f(X)} \tag{2}$$

Let's understand each term:

- $\mathbf{P(\theta|X)}$ — **Posterior Probability**
  - The probability distribution of θ **after** seeing the data X
  - This is what we want to find—our updated belief about θ
  - Read as: "the probability of theta **given** the data"
- $\mathbf{P(X|\theta)}$ — **Likelihood**
  - The probability of observing data X **if** the parameter were θ
  - This is the same likelihood we've seen in MLE!
  - Read as: "the probability of the data given theta"
- $\mathbf{P(\theta)}$ — **Prior Probability**
  - The probability distribution of θ **before** seeing any data
  - Represents our initial belief or background knowledge
  - Read as: "the prior probability of theta"
- $\mathbf{P(X)}$ — **Evidence (Marginal Likelihood)**
  - The total probability of observing the data across all possible values of θ
  - Computed as $P(X) = \int P(X|\theta)P(\theta)d\theta$
  - Acts as a **normalizing constant** to ensure probabilities sum to 1

2.2 The Proportionality Form

Since $P(X)$ doesn't depend on θ, we often write:

Key Summary

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \tag{3}$$

$$f(\theta|X) \propto f(X|\theta) \cdot f(\theta) \tag{4}$$

The posterior is **proportional to** the likelihood times the prior. We can always normalize later to get a proper probability distribution.

Example: Intuitive Interpretation

Think of Bayes' Rule as a belief-updating mechanism:

1. **Start with your prior belief** $P(\theta)$: What you thought before seeing any data
2. **Observe data and compute likelihood** $P(X|\theta)$: How likely is this data under different values of θ?
3. **Update your belief** $P(\theta|X)$: Combine prior and likelihood to get your new, informed belief

Values of θ that:

- Were believed likely (high prior) AND make the data likely (high likelihood) $\rightarrow$ stay high in posterior
- Were believed unlikely (low prior) OR make the data unlikely (low likelihood) $\rightarrow$ low in posterior

3 A Discrete Example: Coin Selection

Let's work through a complete example to solidify these concepts.

3.1 Problem Setup

You have three coins in your pocket:

- **Coin A:** Biased toward tails with $P(\text{heads}) = 0.1$
- **Coin B:** Fair coin with $P(\text{heads}) = 0.5$
- **Coin C:** Biased toward heads with $P(\text{heads}) = 0.9$

You randomly pull out one coin and flip it 4 times, getting **3 heads and 1 tail**.

Question: What's the probability that you drew each coin?

3.2 Step-by-Step Solution

Step 1: Define the Prior Distribution

Before flipping the coin, you have no information about which coin was selected. Since you drew randomly from three coins:

$$P(\theta = 0.1) = P(\theta = 0.5) = P(\theta = 0.9) = \frac{1}{3} \tag{5}$$

This is a **discrete uniform prior** over the three possible values.

Step 2: Define the Likelihood (Data Model)

The number of heads in 4 flips follows a **Binomial distribution**:

$$P(X = k|\theta) = \binom{4}{k}\theta^k(1-\theta)^{4-k} \tag{6}$$

We observed $X = 3$ (3 heads out of 4 flips).

Step 3: Calculate the Likelihood for Each Coin

For each possible value of θ, calculate $P(X = 3|\theta)$:

$$P(X = 3|\theta = 0.1) = \binom{4}{3}(0.1)^3(0.9)^1 = 4 \times 0.001 \times 0.9 = \mathbf{0.0036} \tag{7}$$

$$P(X = 3|\theta = 0.5) = \binom{4}{3}(0.5)^3(0.5)^1 = 4 \times 0.125 \times 0.5 = \mathbf{0.25} \tag{8}$$

$$P(X = 3|\theta = 0.9) = \binom{4}{3}(0.9)^3(0.1)^1 = 4 \times 0.729 \times 0.1 = \mathbf{0.2916} \tag{9}$$

Observation: Coin C ($\theta = 0.9$) makes the data most likely, which makes sense—if a coin is biased toward heads, we're more likely to see 3 heads!

Step 4: Apply Bayes' Rule

Calculate the unnormalized posterior for each value:

$$P(\theta = 0.1|X = 3) \propto 0.0036 \times \frac{1}{3} = 0.0012 \tag{10}$$

$$P(\theta = 0.5|X = 3) \propto 0.25 \times \frac{1}{3} = 0.0833 \tag{11}$$

$$P(\theta = 0.9|X = 3) \propto 0.2916 \times \frac{1}{3} = 0.0972 \tag{12}$$

Step 5: Normalize to Get the Posterior

Sum the unnormalized values: $0.0012 + 0.0833 + 0.0972 = 0.1817$

Divide each by this sum:

$$P(\theta = 0.1|X = 3) = \frac{0.0012}{0.1817} \approx \mathbf{0.7}\% \tag{13}$$

$$P(\theta = 0.5|X = 3) = \frac{0.0833}{0.1817} \approx \mathbf{45.8}\% \tag{14}$$

$$P(\theta = 0.9|X = 3) = \frac{0.0972}{0.1817} \approx \mathbf{53.5}\% \tag{15}$$

Key Summary

Results:

Coin	Prior	Likelihood	Posterior
A ($\theta = 0.1$)	33.3%	0.0036	**0.7%**
B ($\theta = 0.5$)	33.3%	0.2500	**45.8%**
C ($\theta = 0.9$)	33.3%	0.2916	**53.5%**

Interpretation: Before seeing the data, each coin was equally likely (33.3% each). After observing 3 heads in 4 flips, we now believe Coin C (biased toward heads) is most likely (53.5%), Coin B (fair) is also plausible (45.8%), and Coin A (biased toward tails) is nearly ruled out (0.7%).

3.3 The Effect of Sample Size

Important Note

What happens if we flip more coins?

If instead of 4 flips with 3 heads, we had **40 flips with 30 heads** (same proportion: 75%):

- The likelihood for Coin A ($\theta = 0.1$) would be **astronomically small** (essentially 0)
- The likelihoods for Coins B and C would shift even more toward C
- The posterior would show: A $\approx$ 0%, B $\approx$ 35%, C $\approx$ 65%

Key Insight: With more data, the posterior becomes more concentrated. The data "speaks louder" than the prior as sample size increases.

4 Choosing Prior Distributions

One of the most important (and sometimes controversial) aspects of Bayesian modeling is choosing the **prior distribution**. There are three main approaches:

4.1 Informative Priors

Definition: Informative Prior

A prior distribution that encodes specific knowledge or beliefs about the parameter before seeing data.

When to use:

- You have expert knowledge about reasonable parameter values
- You have data from previous studies
- Physical or logical constraints exist on the parameter

Example: Temperature Prediction

Predicting tomorrow's noon temperature in Boston during October:

Prior Information:

- Historical average for this date: 60°F
- Standard deviation of day-to-day variation: 5°F

Informative Prior:

$$\mu \sim \text{Normal}(\mu_0 = 60, \sigma_0^2 = 25) \tag{16}$$

This says: "I believe the temperature will be around 60°F, probably within 50–70°F (95% of the time within ±2 standard deviations)."

4.2 Uninformative (Vague) Priors

Definition: Uninformative Prior

A prior distribution designed to have minimal influence on the posterior, letting the data dominate the inference.

When to use:

- You have no prior knowledge about the parameter
- You want to be "objective" and let data speak for itself
- You're concerned about prior sensitivity

Example: Success Probability

Estimating the success probability p of a new medical treatment:

Uninformative Prior:

$$p \sim \text{Uniform}(0, 1) \tag{17}$$

This says: "I have no idea what p is. Any value between 0 and 1 is equally plausible before seeing the data."

4.3 Conjugate Priors

Definition: Conjugate Prior

A prior distribution that, when combined with a particular likelihood function, produces a posterior distribution of the **same family** as the prior.

Why use conjugate priors?

- **Mathematical convenience:** The posterior has a known closed-form solution
- **Interpretability:** Prior parameters often have intuitive meanings
- **Computational efficiency:** No need for numerical integration or simulation

Key Summary

Common Conjugate Prior-Likelihood Pairs:

Likelihood	Parameter	Conjugate Prior	Posterior
Binomial	p (success probability)	Beta	Beta
Poisson	λ (rate)	Gamma	Gamma
Normal (known σ^2)	μ (mean)	Normal	Normal
Normal	$1/\sigma^2$ (precision)	Gamma	Gamma
Exponential	λ (rate)	Gamma	Gamma

5 The Normal-Normal Model

The most important conjugate model for continuous data is the **Normal-Normal model**, where both the data likelihood and the prior on the mean are normal distributions.

5.1 Model Setup

Likelihood (Data Model):

$$X_1, X_2, \ldots, X_n \sim \text{Normal}(\mu, \sigma^2) \quad \text{(i.i.d.)} \tag{18}$$

- μ is the unknown population mean (the parameter we want to estimate)
- σ^2 is the known variance (we assume this is fixed and known for simplicity)

Prior Distribution:

$$\mu \sim \text{Normal}(\mu_0, \sigma_0^2) \tag{19}$$

- μ_0 is the **prior mean** — our best guess for μ before seeing data
- σ_0^2 is the **prior variance** — how uncertain we are about this guess
- These are called **hyperparameters** (parameters of the prior distribution)

5.2 The Posterior Distribution

Because Normal-Normal is a conjugate pair, the posterior is also Normal:

$$\mu|X \sim \text{Normal}(\mu_n, \sigma_n^2) \tag{20}$$

where the **posterior mean** is:

$$\mu_n = \frac{\sigma^2 \cdot \mu_0 + n \cdot \sigma_0^2 \cdot \bar{X}}{\sigma^2 + n \cdot \sigma_0^2} \tag{21}$$

and the **posterior variance** is:

$$\sigma_n^2 = \frac{\sigma^2 \cdot \sigma_0^2}{\sigma^2 + n \cdot \sigma_0^2} \tag{22}$$

5.3 Understanding the Posterior

Let's dissect these formulas to understand what they're telling us:

Key Information

Posterior Mean as a Weighted Average:
The posterior mean μ_n can be rewritten as:

$$\mu_n = w \cdot \mu_0 + (1 - w) \cdot \bar{X} \tag{23}$$

where $w = \frac{\sigma^2}{\sigma^2 + n \cdot \sigma_0^2}$

This is a **weighted average** of:

- μ_0: the prior mean (what we believed before)
- $\bar{X}$: the sample mean (what the data says)

The weight w depends on:

- **Sample size** n: More data → smaller w → more weight on $\bar{X}$
- **Prior uncertainty** σ_0^2: Larger σ_0^2 → smaller w → more weight on $\bar{X}$
- **Data variance** σ^2: Larger σ^2 → larger w → more weight on μ_0

Important Note

Limiting Cases:

1. **As** $n \to \infty$ (lots of data):
   - $\mu_n \to \bar{X}$ (posterior mean approaches sample mean)
   - $\sigma_n^2 \to 0$ (posterior variance shrinks to zero)
   - **Data dominates**: With enough data, the prior becomes irrelevant
2. **As** $\sigma_0^2 \to 0$ (very confident prior):
   - $\mu_n \to \mu_0$ (posterior stays at prior mean)
   - **Prior dominates**: A very strong prior can't be overridden by data
3. **As** $\sigma_0^2 \to \infty$ (uninformative prior):
   - $\mu_n \to \bar{X}$ (posterior mean equals sample mean)
   - $\sigma_n^2 \to \sigma^2/n$ (same as frequentist standard error!)
   - **Data dominates**: Bayesian results match frequentist results

Example: Numerical Example

Setup:

- Prior: $\mu \sim \text{Normal}(\mu_0 = 100, \sigma_0^2 = 25)$
- Data: $n = 10$ observations with $\bar{X} = 110$, known $\sigma^2 = 100$

Calculate Posterior:

$$\mu_n = \frac{100 \cdot 100 + 10 \cdot 25 \cdot 110}{100 + 10 \cdot 25} = \frac{10000 + 27500}{350} = \frac{37500}{350} = 107.14 \tag{24}$$

$$\sigma_n^2 = \frac{100 \cdot 25}{100 + 250} = \frac{2500}{350} = 7.14 \tag{25}$$

Result: $\mu|X \sim \text{Normal}(107.14, 7.14)$

Interpretation:

- Prior mean was 100, data mean was 110
- Posterior mean (107.14) is pulled toward the data but not all the way
- Posterior standard deviation is $\sqrt{7.14} \approx 2.67$
- 95% Credible Interval: $107.14 \pm 1.96 \times 2.67 = [101.9, 112.4]$

6 Bayesian Point and Interval Estimation

Once we have the posterior distribution, we can extract useful summaries.

6.1 Point Estimates

The posterior is a full distribution, but sometimes we need a single "best guess":

- **Posterior Mean**: $E[\theta|X]$
  - The expected value of the posterior distribution
  - Minimizes squared error loss
  - Most commonly used
- **Posterior Mode (MAP)**: $\arg\max_\theta f(\theta|X)$
  - The value where the posterior is highest
  - Called **Maximum A Posteriori (MAP)** estimate
  - Connects to Ridge and Lasso (more on this later!)
- **Posterior Median**: The 50th percentile
  - The value that splits the posterior in half
  - Minimizes absolute error loss

For a Normal posterior, all three are equal (because the Normal is symmetric).

6.2 Credible Intervals

Definition: Credible Interval

A $100(1-\alpha)\%$ **credible interval** is a range $[a, b]$ such that:

$$P(a \leq \theta \leq b|X) = 1 - \alpha \tag{26}$$

For a Normal posterior $\mu|X \sim \text{Normal}(\mu_n, \sigma_n^2)$:

$$95\% \text{ Credible Interval} = \mu_n \pm 1.96 \times \sigma_n \tag{27}$$

Critical: Credible vs. Confidence Intervals

This is one of the most important distinctions in statistics!

Bayesian 95% Credible Interval:

- **Interpretation:** "Given the data we observed, there is a 95% probability that θ lies in this interval."
- The parameter θ is random, the interval is fixed (once calculated)
- **This is the intuitive interpretation most people want!**

Frequentist 95% Confidence Interval:

- **Interpretation:** "If we repeated this experiment many times and computed a confidence interval each time, 95% of those intervals would contain the true θ."

- The parameter θ is fixed, the interval is random (varies across experiments)
- **Warning:** You CANNOT say "there's a 95% probability θ is in this interval"

7 Bayesian Linear Regression

We can apply Bayesian principles to linear regression by placing prior distributions on the regression coefficients.

7.1 Model Setup

Likelihood (Same as OLS):

$$y_i \sim \text{Normal}(\beta_0 + \beta_1 x_i, \sigma^2) \tag{28}$$

Unknown Parameters: $\beta_0, \beta_1, \sigma^2$

Prior Distributions (Conjugate):

$$\beta_0 \sim \text{Normal}(\mu_{\beta_0}, \sigma^2_{\beta_0}) \tag{29}$$

$$\beta_1 \sim \text{Normal}(\mu_{\beta_1}, \sigma^2_{\beta_1}) \tag{30}$$

$$1/\sigma^2 \sim \text{Gamma}(a_0, \lambda_0) \tag{31}$$

7.2 What Bayesian Regression Provides

Instead of single point estimates, we get **entire posterior distributions** for each parameter:

- $\beta_0|X, y$ has a posterior distribution
- $\beta_1|X, y$ has a posterior distribution
- $\sigma^2|X, y$ has a posterior distribution

This allows us to make probabilistic statements like:

- "There's a 97% probability that $\beta_1 > 0$"
- "The 95% credible interval for β_1 is [2.1, 4.8]"
- "Given the data, there's less than 5% chance that $|\beta_1| > 10$"

8 Connection to Ridge and Lasso

One of the most beautiful insights in statistics is the connection between Bayesian inference and regularization methods like Ridge and Lasso.

8.1 Recall: Regularized Loss Functions

Ordinary Least Squares (OLS):

$$\text{Loss}_{OLS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{32}$$

Ridge Regression (L2 penalty):

$$\text{Loss}_{Ridge} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{33}$$

Lasso Regression (L1 penalty):

$$\text{Loss}_{Lasso} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{34}$$

8.2 The MAP Connection

In Bayesian inference, the **MAP (Maximum A Posteriori)** estimate is:

$$\hat{\beta}_{MAP} = \arg\max_{\beta} f(\beta|X) = \arg\max_{\beta}[f(X|\beta) \cdot f(\beta)] \tag{35}$$

Taking the logarithm (which preserves the maximum):

$$\hat{\beta}_{MAP} = \arg\max_{\beta}[\log f(X|\beta) + \log f(\beta)] \tag{36}$$

Equivalently, minimizing the negative:

$$\hat{\beta}_{MAP} = \arg\min_{\beta}[-\log f(X|\beta) - \log f(\beta)] \tag{37}$$

Key observation:

- $-\log f(X|\beta)$ is proportional to the **sum of squared errors** (for Normal likelihood)
- $-\log f(\beta)$ is the **penalty term** from the prior!

8.3 Ridge = Normal Prior

Definition: Ridge Regression as Bayesian MAP

If we place a **Normal prior centered at zero** on each β_j:

$$\beta_j \sim \text{Normal}(0, \tau^2) \tag{38}$$

Then:

$$-\log f(\beta_j) = -\log\left(\frac{1}{\sqrt{2\pi\tau^2}} e^{-\beta_j^2/2\tau^2}\right) = \text{constant} + \frac{\beta_j^2}{2\tau^2} \tag{39}$$

This is proportional to β_j^2, the **L2 penalty**!

Key Summary

Ridge Regression is Bayesian MAP with Normal Prior

- The penalty $\lambda \sum \beta_j^2$ comes from assuming $\beta_j \sim N(0, \tau^2)$
- The regularization parameter λ is inversely related to prior variance τ^2
- Larger λ = smaller prior variance = stronger belief that β_j should be near zero

Intuition: The Normal prior says "I believe coefficients are probably small (near zero) with a smooth, bell-shaped probability." This pulls estimates toward zero but doesn't make them exactly zero.

8.4 Lasso = Laplace Prior

Definition: Lasso Regression as Bayesian MAP

If we place a **Laplace (Double Exponential) prior** on each β_j:

$$f(\beta_j) = \frac{1}{2b} e^{-|\beta_j|/b} \tag{40}$$

Then:

$$-\log f(\beta_j) = \text{constant} + \frac{|\beta_j|}{b} \tag{41}$$

This is proportional to $|\beta_j|$, the **L1 penalty**!

Key Summary

Lasso Regression is Bayesian MAP with Laplace Prior

- The penalty $\lambda \sum |\beta_j|$ comes from assuming a Laplace prior on β_j
- The Laplace distribution has a **sharp peak at zero**
- This creates **sparse solutions** where some $\beta_j = 0$ exactly

Intuition: The Laplace prior says "I strongly believe most coefficients should be exactly zero." The sharp peak at zero gives high probability mass to $\beta_j = 0$, encouraging sparsity (feature selection).

Important Note

Visual Comparison: Normal vs. Laplace Priors

Normal Distribution (Ridge):

- Smooth, bell-shaped curve
- Gradual decrease away from zero
- Coefficients are pulled toward zero but rarely exactly zero

Laplace Distribution (Lasso):

- Sharp, peaked curve at zero
- Rapid decrease away from zero
- High probability mass at exactly zero → sparse solutions

8.5 Summary: Regularization as Bayesian Inference

Key Information

Method	Penalty	Bayesian Prior	Effect
OLS	None	Flat/Improper	No shrinkage
Ridge	$\lambda \sum \beta_j^2$	Normal$(0, \tau^2)$	Shrink toward zero
Lasso	$\lambda \sum \lvert\beta_j\rvert$	Laplace$(0, b)$	Shrink + Sparsity

Key Insight: The choice of regularization is implicitly a choice of prior belief about the coefficients!

9 Computational Methods: MCMC

When conjugate priors don't apply or models become complex, we need computational methods to approximate the posterior distribution.

9.1 The Problem

The posterior is:

$$f(\theta|X) = \frac{f(X|\theta)f(\theta)}{f(X)} = \frac{f(X|\theta)f(\theta)}{\int f(X|\theta')f(\theta')d\theta'} \tag{42}$$

The integral in the denominator is often **intractable** (impossible to compute analytically).

9.2 The Solution: Simulation

Instead of computing the posterior exactly, we **draw samples** from it:

$$\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)} \sim f(\theta|X) \tag{43}$$

With enough samples, we can approximate any property of the posterior:

- Posterior mean: $E[\theta|X] \approx \frac{1}{N}\sum_{i=1}^{N}\theta^{(i)}$
- Posterior variance: $\text{Var}[\theta|X] \approx \frac{1}{N-1}\sum_{i=1}^{N}(\theta^{(i)} - \bar{\theta})^2$
- Credible interval: Sort samples and find the 2.5th and 97.5th percentiles

9.3 MCMC: Markov Chain Monte Carlo

Definition: MCMC

Markov Chain Monte Carlo is a class of algorithms that generate samples from a target distribution by constructing a Markov chain whose stationary distribution is the target.

Intuition: Imagine a random walker exploring a landscape (the posterior distribution). The walker:

- Tends to stay in high-probability regions (peaks)
- Occasionally visits low-probability regions (valleys)
- After walking long enough, the proportion of time spent in each region matches the posterior probability

9.4 Gibbs Sampling

For multi-parameter models, **Gibbs Sampling** is a popular MCMC algorithm:

Idea: Instead of sampling all parameters at once, sample each parameter **one at a time**, conditional on the current values of the others.

Algorithm (for 3 parameters $\theta_1, \theta_2, \theta_3$):

1. Initialize: $\theta_1^{(0)}, \theta_2^{(0)}, \theta_3^{(0)}$

2. For $t = 1, 2, \ldots, N$:
   - 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. Discard early samples (burn-in period) to remove initialization effects
4. Use remaining samples as draws from the joint posterior

Important Note

Burn-in Period

The initial samples depend heavily on where we started (the initialization). We discard the first several thousand samples (the "burn-in" period) to ensure the remaining samples are representative of the posterior.

10 Quiz Review: Categorical Variables

Before diving into Bayesian modeling, let's review an important concept from the quiz about interpreting categorical (dummy) variables in regression.

10.1 Setup

Consider predicting hours spent on homework based on course registration (4 groups):

- AC 209 (baseline/reference group)
- CS 1090
- CSCI E-109
- STAT 109

The model produces:

$$\hat{y} = 11.0 - 2.0 \cdot x_{CS1090} + 3.5 \cdot x_{CSCIE109} + 5.0 \cdot x_{STAT109} \tag{44}$$

10.2 Interpretation

- **Intercept (11.0):** The predicted hours for the **baseline group** (AC 209). This is also the sample mean for AC 209 students.
- **Coefficient for CS 1090 (-2.0):** CS 1090 students spend, on average, **2 fewer hours** than AC 209 students. Predicted hours: $11.0 - 2.0 = 9.0$
- **Coefficient for CSCI E-109 (+3.5):** CSCI E-109 students spend **3.5 more hours** than AC 209. Predicted hours: $11.0 + 3.5 = 14.5$
- **Coefficient for STAT 109 (+5.0):** STAT 109 students spend **5 more hours** than AC 209. Predicted hours: $11.0 + 5.0 = 16.0$

Important Note

Common Mistake: "Controlling for other variables"
When the only predictor is a categorical variable (like course registration), do NOT say "controlling for other variables" in your interpretation. There are no other variables to control for! Each coefficient simply represents the difference from the baseline group.

10.3 Changing the Baseline Group

If CS 1090 were the baseline instead of AC 209:

- New intercept: $11.0 - 2.0 = 9.0$ (mean for CS 1090)
- New coefficient for AC 209: +2.0 (now positive, since AC 209 is 2 hours above CS 1090)
- Other coefficients adjust accordingly

11 Quiz Review: Validation Metrics

11.1 The Question

When using Ridge regression with cross-validation, which metric should we use to evaluate model performance on the validation set?

Option A: MSE (Mean Squared Error)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{45}$$

Option B: Ridge loss with penalty

$$\text{Ridge Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \tag{46}$$

11.2 The Answer: Option A (MSE)

Key Summary

Why use MSE without the penalty for validation?

1. **Training vs. Validation have different purposes:**
   - **Training:** Find coefficients β that balance fit and complexity (use penalty)
   - **Validation:** Assess how well the model predicts new data (use natural error metric)
2. **The penalty is part of the optimization process, not the evaluation:**
   - The penalty term $\lambda\sum\beta_j^2$ is a regularization tool to prevent overfitting
   - It's not a measure of predictive accuracy
3. **We want to compare models fairly:**
   - Different λ values lead to different penalties
   - Using penalized loss would unfairly favor models with larger λ
   - MSE measures true prediction error, which is what we care about

12 Chapter Summary

Key Summary

Key Concepts from Lecture 11:

1. Bayesian vs. Frequentist Thinking

- Frequentist: Parameters are fixed, data is random
- Bayesian: Parameters have distributions (representing uncertainty), data is fixed once observed

2. Bayes' Rule

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \tag{47}$$

$$f(\theta|X) \propto f(X|\theta) \cdot f(\theta) \tag{48}$$

3. Types of Priors

- **Informative:** Encode specific prior knowledge
- **Uninformative:** Let data dominate
- **Conjugate:** Mathematical convenience (posterior same family as prior)

4. Normal-Normal Model

- Posterior mean is weighted average of prior mean and sample mean
- With more data, posterior concentrates around sample mean
- With stronger prior, posterior stays near prior mean

5. Credible vs. Confidence Intervals

- Credible: "95% probability parameter is in this interval" (Bayesian, intuitive)
- Confidence: "95% of intervals from repeated experiments contain true value" (Frequentist)

6. Regularization as Bayesian MAP

- **Ridge = Normal prior** on coefficients (shrinkage)
- **Lasso = Laplace prior** on coefficients (shrinkage + sparsity)

7. MCMC for Complex Models

- Sample from posterior when analytical solutions don't exist
- Gibbs sampling: Sample parameters one at a time
- Discard burn-in samples, use remaining samples for inference

13 Key Formulas Reference

Key Information

Bayes' Rule:

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)} \quad \text{or} \quad f(\theta|X) \propto f(X|\theta)f(\theta) \tag{49}$$

Normal-Normal Posterior:

$$\mu_n = \frac{\sigma^2\mu_0 + n\sigma_0^2\bar{X}}{\sigma^2 + n\sigma_0^2}, \quad \sigma_n^2 = \frac{\sigma^2\sigma_0^2}{\sigma^2 + n\sigma_0^2} \tag{50}$$

95% Credible Interval (Normal):

$$[\mu_n - 1.96\sigma_n, \quad \mu_n + 1.96\sigma_n] \tag{51}$$

MAP Estimation:

$$\hat{\theta}_{MAP} = \arg\max_{\theta} f(\theta|X) = \arg\max_{\theta}[f(X|\theta)f(\theta)] \tag{52}$$

Ridge = Normal Prior:

$$\beta_j \sim N(0, \tau^2) \quad \Rightarrow \quad \text{Penalty} = \lambda \sum \beta_j^2 \tag{53}$$

Lasso = Laplace Prior:

$$\beta_j \sim \text{Laplace}(0, b) \quad \Rightarrow \quad \text{Penalty} = \lambda \sum |\beta_j| \tag{54}$$

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):

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

Lecture 13: Classification and Logistic Regression

CS109A: Introduction to Data Science

Harvard University

Course:	CS109A: Introduction to Data Science
Lecture:	Lecture 13
Instructors:	Pavlos Protopapas, Kevin Rader, Chris Gumb
Topics:	Classification vs. Regression, Why linear regression fails for classification, Logistic Regression, Sigmoid function, Odds and Log-Odds interpretation, Maximum Likelihood Estimation, Binary Cross-Entropy, Decision Boundaries

Contents

Lecture Overview

This lecture marks a major transition in the course: we move from **regression** problems (predicting numeric values) to **classification** problems (predicting categories).

Key Learning Objectives:

- Understand the fundamental difference between regression and classification
- Learn why linear regression is unsuitable for classification tasks
- Master logistic regression: the foundational parametric model for classification
- Interpret logistic regression coefficients in terms of odds and log-odds
- Understand the probabilistic foundation: Bernoulli likelihood and MLE
- Visualize and understand decision boundaries

This lecture also includes a review section covering hypothesis testing, permutation tests, and interaction terms for midterm preparation.

1.1 Regression vs. Classification

Definition: Regression Problems

In **regression**, the response variable Y is **quantitative** (numeric, continuous).

Examples:

- Predicting house prices ($500,000)
- Predicting tomorrow's temperature (72.5°F)
- Predicting sales revenue ($1.2 million)

Models: Linear regression, polynomial regression, Ridge, Lasso, k-NN (for regression)

Definition: Classification Problems

In **classification**, the response variable Y is **qualitative** (categorical).

Examples:

- Predicting whether an email is spam or not spam (binary)
- Predicting whether a patient has heart disease (Yes/No)
- Predicting which major a student will choose (CS/Stats/Other)
- Predicting handwritten digits (0-9)

Models: Logistic regression, k-NN (for classification), decision trees, neural networks

Example: Regression vs. Classification Question

Consider a medical dataset with patient information:

Regression question: "What will this patient's maximum heart rate be?"

- Answer: A number (e.g., 152 bpm)

Classification question: "Does this patient have heart disease?"

- Answer: A category (Yes or No)

2 Why Not Linear Regression for Classification?

A natural first thought might be: "Can't we just use linear regression for classification by encoding categories as numbers?" Let's see why this fails.

2.1 Problem 1: Multi-class Encoding Creates False Ordering

Suppose we want to predict a student's major with three categories: CS, Statistics, and Other. We might encode these as:

- $Y = 1$ if CS
- $Y = 2$ if Statistics
- $Y = 3$ if Other

Important Note

The Problem: Linear regression assumes numerical relationships between Y values!
When we fit a linear regression, the model interprets:

- The "distance" from CS (1) to Statistics (2) is the same as from Statistics (2) to Other (3)
- A one-unit change in X corresponds to moving one "step" on this scale

This is **completely meaningless** for categorical data! The categories have no inherent numerical order or spacing.
If we had encoded differently ($Y = 1$ for Other, $Y = 2$ for CS, $Y = 3$ for Statistics), we'd get a completely different model!

2.2 Problem 2: Binary Predictions Can Exceed [0, 1]

Even with binary classification (only two categories), linear regression fails in a different way.

Consider predicting heart disease (Yes = 1, No = 0) based on maximum heart rate. If we fit a linear regression $Y = \beta_0 + \beta_1 X$, we can interpret $\hat{Y}$ as the "probability" that $Y = 1$.

Important Note

The Problem: Probabilities must be between 0 and 1!
A straight line has no bounds. For extreme values of X:

- **Very low heart rate:** Model might predict $P(\text{heart disease}) = 1.1$ (110%)
- **Very high heart rate:** Model might predict $P(\text{heart disease}) = -0.2$ (-20%)

These predictions are mathematically and logically invalid!

Example: Linear Regression for Heart Disease

If we fit a linear regression to predict heart disease from maximum heart rate, we get:

- A downward sloping line (higher heart rate → lower probability)
- But for patients with very low heart rates (e.g., 50 bpm), the model predicts probabilities > 1
- For very fit athletes with high heart rates (e.g., 200 bpm), the model predicts negative probabilities

We need a function that "squashes" predictions to stay between 0 and 1!

3 Logistic Regression: The Solution

3.1 The Sigmoid Function

Instead of predicting probabilities directly with a line, we use an **S-shaped curve** that is naturally bounded between 0 and 1.

Definition: Sigmoid (Logistic) Function

The **sigmoid function** maps any real number to a value between 0 and 1:

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z} \tag{1}$$

Properties:

- As $z \to +\infty$: $\sigma(z) \to 1$
- As $z \to -\infty$: $\sigma(z) \to 0$
- At $z = 0$: $\sigma(0) = 0.5$
- The function is **monotonically increasing**
- Output is always in $(0, 1)$

3.2 The Logistic Regression Model

In logistic regression, we model the **probability** that $Y = 1$:

$$P(Y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}} \tag{2}$$

Or equivalently:

$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \tag{3}$$

Key Summary

Logistic Regression in Two Steps:

1. **Linear part (same as linear regression):** Compute $h = \beta_0 + \beta_1 X$
2. **Sigmoid transformation:** Pass h through the sigmoid to get probability $p = \sigma(h)$

The sigmoid "squashes" the linear prediction to always be between 0 and 1.

3.3 Understanding the Shape

The parameters β_0 and β_1 control the shape of the S-curve:

- β_0 (intercept): Controls the **horizontal position** of the curve
  - Larger β_0 shifts the curve to the left (higher probabilities for the same X)
  - Smaller β_0 shifts the curve to the right
- β_1 (slope): Controls the **steepness** of the curve

- Larger $|\beta_1|$ makes the transition from 0 to 1 sharper
- Smaller $|\beta_1|$ makes the transition more gradual
- Positive β_1: Probability increases as X increases
- Negative β_1: Probability decreases as X increases

4 Interpreting Logistic Regression: Odds and Log-Odds

4.1 The Problem with Direct Interpretation

With linear regression, interpretation is simple: "A one-unit increase in X is associated with a β_1 change in Y."

With logistic regression, the relationship between X and $P(Y = 1)$ is non-linear (S-shaped), so this simple interpretation doesn't work.

4.2 Odds

Definition: Odds

The **odds** of an event is the ratio of the probability of success to the probability of failure:

$$\text{Odds} = \frac{p}{1-p} \tag{4}$$

where $p = P(\text{success})$.

Examples:

- If $p = 0.5$ (50% chance): Odds $= \frac{0.5}{0.5} = 1$ (1:1, or "even odds")
- If $p = 0.8$ (80% chance): Odds $= \frac{0.8}{0.2} = 4$ (4:1, "4 to 1")
- If $p = 0.2$ (20% chance): Odds $= \frac{0.2}{0.8} = 0.25$ (1:4)

Intuition: Odds tell you "how many times more likely is success than failure?"

4.3 Log-Odds (Logit)

Definition: Log-Odds / Logit

The **log-odds** (also called **logit**) is the natural logarithm of the odds:

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) \tag{5}$$

Key properties:

- Log-odds can be any real number ($-\infty$ to $+\infty$)
- When $p = 0.5$: log-odds $= \ln(1) = 0$
- When $p > 0.5$: log-odds > 0
- When $p < 0.5$: log-odds < 0

4.4 The Log-Odds Formulation

Here's the key insight: if we solve the logistic regression equation for the linear part, we get:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X \tag{6}$$

Key Summary

Logistic Regression Models Log-Odds Linearly!
The log-odds of success is a **linear function** of the predictors. This is why logistic regression is still considered a "linear" model.

4.5 Interpreting Coefficients

Key Information

β_1 Interpretation (Log-Odds):
A one-unit increase in X is associated with a β_1 **additive change** in the log-odds of $Y = 1$.
e^{β_1} Interpretation (Odds Ratio):
A one-unit increase in X is associated with the odds being **multiplied by** e^{β_1}.

Example: Heart Disease Model

Suppose we fit a logistic regression to predict heart disease from maximum heart rate and get:

$$\ln\left(\frac{P(\text{HeartDisease})}{1-P(\text{HeartDisease})}\right) = 6.325 - 0.0434 \times \text{MaxHR} \tag{7}$$

Interpreting $\beta_1 = -0.0434$:
Log-odds interpretation:

- For each 1 bpm increase in maximum heart rate, the log-odds of heart disease **decrease** by 0.0434.

Odds ratio interpretation:

- $e^{-0.0434} \approx 0.957$
- For each 1 bpm increase in maximum heart rate, the odds of heart disease are **multiplied by 0.957** (i.e., decrease by about 4.3%).

Intuition: Higher maximum heart rate is associated with lower risk of heart disease.

Important Note

Special Values of β_1:

- $\beta_1 = 0$: $e^0 = 1$ (odds multiplied by 1 = no change) $\rightarrow$ **No association**
- $\beta_1 > 0$: $e^{\beta_1} > 1$ (odds increase) $\rightarrow$ **Positive association**
- $\beta_1 < 0$: $e^{\beta_1} < 1$ (odds decrease) $\rightarrow$ **Negative association**

5 Estimating Logistic Regression: MLE

5.1 The Probabilistic Perspective

In linear regression, we assumed errors were normally distributed and minimized MSE (equivalent to maximizing Gaussian likelihood).

In logistic regression, our outcomes are binary (0 or 1), so we use a different distribution.

Definition: Bernoulli Distribution

A **Bernoulli random variable** Y takes value 1 with probability p and value 0 with probability $1-p$:

$$P(Y = y) = p^y(1-p)^{1-y} \tag{8}$$

This is like a single coin flip with probability p of heads.

5.2 Likelihood Function

For logistic regression, we assume each observation y_i comes from a Bernoulli distribution with probability p_i that depends on X_i:

$$p_i = P(Y_i = 1|X_i) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_i)}} \tag{9}$$

The **likelihood** of observing our entire dataset is:

$$L(\beta) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} \tag{10}$$

Intuition:

- If $y_i = 1$ (actual success), we want p_i to be high
- If $y_i = 0$ (actual failure), we want $1 - p_i$ to be high
- Good parameters make observed successes have high predicted probabilities

5.3 Maximum Likelihood Estimation

Definition: Maximum Likelihood Estimation (MLE)

MLE finds the parameter values $\hat{\beta}$ that **maximize** the likelihood of observing the data we actually observed:

$$\hat{\beta}_{MLE} = \arg\max_{\beta} L(\beta) \tag{11}$$

5.4 Log-Likelihood and Binary Cross-Entropy

Products are hard to work with, so we take the logarithm:

$$\ell(\beta) = \log L(\beta) = \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \tag{12}$$

Since computers typically **minimize** functions, we minimize the **negative** log-likelihood:

Definition: Binary Cross-Entropy Loss

The **binary cross-entropy** (BCE) loss is the negative log-likelihood:

$$\text{BCE} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \tag{13}$$

This is the loss function that logistic regression minimizes.

Key Summary

Comparison: Linear vs. Logistic Regression

	Linear Regression	**Logistic Regression**
Response type	Continuous	Binary (0/1)
Distribution assumption	Normal	Bernoulli
Loss function	MSE	Binary Cross-Entropy
Estimation method	Closed-form OLS	Iterative optimization (MLE)

Important Note

Why No σ^2 Parameter?
In linear regression, we estimate β_0, β_1, and σ^2 (error variance).
In logistic regression, we only estimate β_0 and β_1. Why?
Answer: The variance is determined by the Bernoulli distribution itself! If $P(Y = 1) = p$, then $\text{Var}(Y) = p(1-p)$. There's no separate "noise" parameter—the randomness comes from the binary nature of the outcome.

6 Multiple Logistic Regression and Decision Boundaries

6.1 Multiple Logistic Regression

Just like linear regression extends to multiple predictors, so does logistic regression:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \tag{14}$$

Interpretation: e^{β_j} is the multiplicative change in odds for a one-unit increase in X_j, **holding all other predictors constant**.

6.2 Making Predictions: From Probability to Class

Logistic regression outputs a **probability**. To make a classification decision, we need a **threshold**:

- Default threshold: $t = 0.5$
- If $P(Y = 1|X) \geq t$: Predict class 1
- If $P(Y = 1|X) < t$: Predict class 0

6.3 Decision Boundaries

Definition: Decision Boundary

The **decision boundary** is the set of points where the model is exactly undecided—where $P(Y = 1) = 0.5$.
For logistic regression, $P(Y = 1) = 0.5$ when the log-odds equal 0:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{15}$$

6.4 Linear vs. Non-linear Boundaries

Linear boundary (default):

With basic logistic regression, the decision boundary is a **linear equation** in the features. In 2D (X_1, X_2), this is a **straight line**.

Non-linear boundary:

To create curved decision boundaries, add **polynomial** or **interaction** terms:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \beta_4 X_1^2 + \beta_5 X_2^2 \tag{16}$$

Setting this equal to 0 gives a **quadratic equation** in X_1 and X_2, which can produce **circles, ellipses, or hyperbolas**.

Important Note

The Model is Still "Linear"!

Even with polynomial features, we still call this a "linear model" because it's linear in the **parameters** (β's).

The trick is that we create non-linear **features** (X_1^2, X_1X_2, etc.) and then do linear regression on those features.

7 Review Section: Hypothesis Testing and Permutation Tests

This section reviews key concepts for the midterm.

7.1 Hypothesis Testing Framework

The Five Steps:

1. **State hypotheses:**
   - H_0: No effect ($\beta_1 = 0$)
   - H_A: There is an effect ($\beta_1 \neq 0$)
2. **Choose test statistic:** Usually $t = \hat{\beta}_1 / SE(\hat{\beta}_1)$
3. **Compute test statistic** from data
4. **Calculate p-value:** Probability of seeing a result this extreme if H_0 is true
5. **Make decision:** Reject H_0 if p-value $< \alpha$ (usually 0.05)

7.2 Permutation Tests

When t-test assumptions (normality, constant variance) are violated:

1. Compute the observed test statistic (e.g., $\hat{\beta}_1 = 0.589$)
2. **Shuffle** Y randomly while keeping X fixed (simulates H_0: no relationship)
3. Fit model to shuffled data, record $\hat{\beta}_1^*$
4. Repeat 1000+ times to build the **null distribution**
5. P-value = proportion of permuted statistics as extreme as observed

7.3 Bootstrap vs. Permutation

	Bootstrap	**Permutation Test**
Purpose	Estimation (confidence intervals)	Hypothesis testing (p-values)
Sampling	With replacement (pairs)	Without replacement (shuffle Y)
Assumption	Data represents population	H_0 is true

7.4 Interaction Terms

In a model like `price ~ sqft * type`:

- **Main effect for sqft:** Effect of sqft for the **reference category**
- **Interaction term (sqft:type):** How the sqft effect **differs** for other categories compared to reference
- **Total effect** for category k: Main effect + interaction term for k

7.5 Confidence vs. Prediction Intervals

- **Confidence interval:** Where we think the **mean** response lies (narrower)
- **Prediction interval:** Where a **new individual** response would lie (wider, includes ϵ)

8 Implementation in Python

8.1 Logistic Regression with sklearn

```python
from sklearn.linear_model import LogisticRegression

# Prepare data
X = df[['MaxHR']]  # Must be 2D array
y = df['HeartDisease']  # Binary: 0 or 1

# Fit logistic regression (no regularization)
logreg = LogisticRegression(penalty=None)  # Note: use None, not 'none'
logreg.fit(X, y)

# Get coefficients
print("beta_1 (coefficient):", logreg.coef_[0][0])
print("beta_0 (intercept):", logreg.intercept_[0])

# Make predictions
probabilities = logreg.predict_proba(X)  # P(Y=0) and P(Y=1)
predictions = logreg.predict(X)  # Class labels (0 or 1)
```

8.2 Permutation Test Implementation

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

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

# Permutation test
n_perms = 1000
perm_betas = []

for _ in range(n_perms):
    # Shuffle Y (breaks any real relationship)
    y_shuffled = np.random.permutation(y)

    # Fit model to shuffled data
    model_perm = LinearRegression().fit(X, y_shuffled)
    perm_betas.append(model_perm.coef_[0])

# P-value: proportion of permuted betas as extreme as observed
p_value = np.mean(np.abs(perm_betas) >= np.abs(observed_beta))
```

9 Key Takeaways

Key Summary

Why Classification Needs Different Methods:

- Linear regression for multi-class creates meaningless numerical ordering
- Linear regression for binary can predict probabilities outside [0, 1]

Logistic Regression:

- Uses sigmoid function to squash predictions to (0, 1)
- Models log-odds as a linear function of features
- Coefficient interpretation: e^{β_1} = multiplicative change in odds

Estimation:

- Assumes Bernoulli distribution for binary outcomes
- Maximizes likelihood (minimizes binary cross-entropy)
- No σ^2 parameter—variance determined by $p(1-p)$

Decision Boundaries:

- Linear model → linear boundary
- Add polynomial/interaction features → curved boundary
- Model is still "linear" in parameters

10 Quick Reference

Key Information

Logistic Regression Model:

$$P(Y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}} \tag{17}$$

Log-Odds Form:

$$\ln\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X \tag{18}$$

Odds Ratio:

$$\text{Odds} = \frac{p}{1-p}, \quad \text{OR} = e^{\beta_1} \tag{19}$$

Binary Cross-Entropy Loss:

$$\text{BCE} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1-y_i)\log(1-p_i)] \tag{20}$$

Decision Boundary (threshold = 0.5):

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots = 0 \tag{21}$$

Lecture 14: Advanced Logistic Regression and Classification Evaluation

CS109A: Introduction to Data Science

Harvard University

■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 14

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Tanner

■ **Topics:** Logistic Regression Inference, Interactions, Decision Boundaries, Regularization, Multiclass Classification, Confusion Matrix, ROC/AUC

Key Summary

This lecture builds upon the foundations of logistic regression to cover advanced topics essential for practical classification tasks.

Key Topics:

- Inference in logistic regression: confidence intervals and hypothesis tests using Z-statistics
- Interpreting coefficients with binary predictors and the concept of reference groups
- Multiple logistic regression with interaction terms
- Decision boundaries: what they are and how polynomial features create non-linear boundaries
- Regularization (Ridge/L2) in logistic regression to prevent overfitting
- Multiclass classification: One-vs-Rest (OvR) and Multinomial approaches
- Classification evaluation: Confusion Matrix, Sensitivity, Specificity, Precision, ROC curves, and AUC

Contents

1 Review: The Probabilistic View of Logistic Regression

1.1 Setting Up the Problem

In logistic regression, our response variable Y takes values 0 or 1 (binary outcomes). We model the **probability of success** $P(Y = 1|X)$ using a probabilistic framework.

Since Y is binary (0 or 1), each observation follows a **Bernoulli distribution**:

$$Y_i|X_i \sim \text{Bernoulli}(p_i)$$

where $p_i = P(Y_i = 1|X_i)$ is the probability of success for observation i.

Definition:

Bernoulli Distribution A random variable Y follows a Bernoulli distribution with parameter p if:

$$P(Y = y) = p^y(1 - p)^{1-y}$$

This gives:

- $P(Y = 1) = p$ (probability of success)
- $P(Y = 0) = 1 - p$ (probability of failure)

The binomial distribution with $n = 1$ is simply a Bernoulli distribution.

1.2 Linking Probability to Predictors

We link the probability p to our predictors through the **log-odds** (logit) function:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X$$

Solving for p, we get the **sigmoid function**:

$$p = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

This is the "exponentiated odds" form mentioned in the lecture.

1.3 The Likelihood Function

Given our data, we want to find the β values that make observing our data most likely. The likelihood function for a single observation is the Bernoulli PMF:

$$L(p|y) = p^y(1 - p)^{1-y}$$

Example:

Understanding the Bernoulli Likelihood Let's verify the likelihood makes intuitive sense:

- If $y = 1$ (success): $L = p^1(1-p)^0 = p$. We want p to be high.
- If $y = 0$ (failure): $L = p^0(1-p)^1 = 1-p$. We want p to be low (so $1-p$ is high).

The formula correctly captures what we want: high probability for what actually happened.

The total likelihood for all n observations is:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

where the β parameters are hidden inside each p_i.

2 Maximum Likelihood Estimation (MLE)

2.1 The Estimation Process

To find the optimal β values, we:

1. Take the **logarithm** of the likelihood (easier to work with sums than products)
2. Take the **derivative** with respect to each β
3. Set the derivatives equal to **zero** and solve

Warning

No Closed-Form Solution!
Unlike linear regression where we can solve for $\hat{\beta} = (X^T X)^{-1} X^T Y$ directly, logistic regression has no closed-form solution. The equations are non-linear and must be solved **numerically**.

2.2 Numerical Optimization

Since we cannot solve analytically, we use numerical methods:

- **Gradient Descent**: Iteratively move in the direction that decreases the loss
- **Newton-Raphson Method**: Uses second derivatives for faster convergence

We typically minimize the **negative log-likelihood**, which is equivalent to maximizing the likelihood. This negative log-likelihood is called **Binary Cross-Entropy**:

$$\text{Loss} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Key Information

What Happens Under the Hood in sklearn
When you call `LogisticRegression().fit(X, y)`, sklearn is running an optimization algorithm (typically LBFGS or similar) to minimize the binary cross-entropy loss and find the optimal β coefficients.

3 Inference in Logistic Regression

Once we have estimates $\hat{\beta}$, we want to:

1. **Interpret** what these coefficients mean
2. **Quantify uncertainty** through confidence intervals
3. **Test hypotheses** about whether coefficients are significantly different from zero

3.1 Z-Statistics vs T-Statistics

3.2 Practical Implications

For confidence intervals:

- Linear regression: $\hat{\beta} \pm t_{\alpha/2,df} \times SE(\hat{\beta})$ (often ≈ 2)
- Logistic regression: $\hat{\beta} \pm \mathbf{1.96} \times SE(\hat{\beta})$ for 95% CI

The value 1.96 comes from the standard normal distribution (Z-distribution).

3.3 Using statsmodels for Inference

While `sklearn` is great for prediction, it does not provide standard errors directly. For statistical inference, use `statsmodels`:

```python
import statsmodels.formula.api as smf

# Fit logistic regression with formula interface
model = smf.logit("heart_disease ~ max_heart_rate", data=df).fit()

# View the summary with coefficients, standard errors, z-stats, p-values, and CIs
print(model.summary())
```

Listing 1: Logistic Regression with statsmodels

The output will include:

- **Coefficients** ($\hat{\beta}$)
- **Standard Errors** (from Fisher's Information)
- **Z-statistic**: $z = \hat{\beta}/SE(\hat{\beta})$
- **P-value**: for testing $H_0 : \beta = 0$
- **95% Confidence Interval**: $\hat{\beta} \pm 1.96 \times SE$

Key Information

Fisher's Information

The standard errors in logistic regression come from **Fisher's Information**, which measures the curvature of the log-likelihood function at its maximum. More curvature (sharper peak) means more certainty, hence smaller standard errors.

4 Interpreting Coefficients with Binary Predictors

When a predictor X is binary (0 or 1), interpretation becomes especially clear.

4.1 Setting Up: Reference Groups

Consider predicting heart disease based on biological sex:

- $X = 1$ if female
- $X = 0$ if male (the **reference group** or **baseline group**)

The model is:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \cdot \text{Female}$$

4.2 Interpreting β_0 (Intercept)

When $X = 0$ (male):

$$\log(\text{Odds}_{\text{male}}) = \beta_0$$

So β_0 is the **log-odds of success for the reference group**.

Example:

Calculating β_0 from Data From the lecture, approximately 52% of males had heart disease:

- Probability: $\hat{p} = 0.52$
- Odds: $\frac{0.52}{1-0.52} = \frac{0.52}{0.48} \approx 1.08$
- Log-odds: $\ln(1.08) \approx 0.077$

So $\beta_0 \approx 0.077$ (or about 0.214 with exact numbers).

4.3 Interpreting β_1 (Slope for Binary Predictor)

β_1 represents the **difference in log-odds** comparing the indicated group (female) to the reference group (male).

When $X = 1$ (female):

$$\log(\text{Odds}_{\text{female}}) = \beta_0 + \beta_1$$

Therefore:

$$\beta_1 = \log(\text{Odds}_{\text{female}}) - \log(\text{Odds}_{\text{male}})$$

Example:

Calculating β_1 from Data From the lecture, approximately 25% of females had heart disease:

- Odds for females: $\frac{0.25}{0.75} = 0.33$
- Log-odds for females: $\ln(0.33) \approx -1.11$
- Difference: $-1.11 - 0.077 \approx -1.19$ (about -1.272 with exact numbers)

A negative β_1 means females have **lower** log-odds of heart disease than males.

4.4 Exponentiation: Odds Ratios

Raw coefficients are on the log-odds scale, which is hard to interpret. **Exponentiate** to get interpretable **odds ratios**:

- e^{β_0} = Odds for the reference group
- e^{β_1} = **Odds Ratio** (multiplicative change in odds)

Definition:

Odds Ratio Interpretation If $e^{\beta_1} = 0.28$:
"The odds of heart disease for females are **0.28 times** (or 28%) the odds for males."
Equivalently: "Females have 72% **lower odds** of heart disease compared to males."

Warning

Odds ≠ Probability!
Never say "72% lower probability." The relationship between odds ratios and probability changes is non-linear and depends on the baseline probability.
Correct: "The odds are 0.28 times as high" or "72% reduction in odds"
Incorrect: "72% reduction in probability"

5 Multiple Logistic Regression

Just like in linear regression, we can include multiple predictors:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

5.1 Key Points

1. **Interpretation**: Each β_j represents the change in log-odds for a one-unit increase in X_j, **holding all other predictors constant**.
2. **Same issues as linear regression**:
   - Multicollinearity (correlated predictors inflate variance)
   - Overfitting (too many predictors relative to sample size)
3. **Visualization**: Instead of fitting an S-curve to a scatter plot, we're fitting an "S-shaped hyperplane" in higher dimensions.

5.2 Interaction Terms

Interaction terms allow the effect of one predictor to **depend on** the value of another predictor.

Example:

Heart Disease: Age × Female Interaction Model:

$$\log(\text{Odds}) = \beta_0 + \beta_1 \cdot \text{Age} + \beta_2 \cdot \text{Female} + \beta_3 \cdot (\text{Age} \times \text{Female})$$

Let's separate this by sex:

For Males (Female = 0):

$$\log(\text{Odds})_{\text{male}} = \beta_0 + \beta_1 \cdot \text{Age}$$

- Intercept: β_0
- Slope for Age: β_1

For Females (Female = 1):

$$\log(\text{Odds})_{\text{female}} = \beta_0 + \beta_1 \cdot \text{Age} + \beta_2 + \beta_3 \cdot \text{Age}$$

$$= (\beta_0 + \beta_2) + (\beta_1 + \beta_3) \cdot \text{Age}$$

- Intercept: $\beta_0 + \beta_2$ (different from males!)
- Slope for Age: $\beta_1 + \beta_3$ (different from males!)

Coefficient Interpretations:

- β_1: Effect of age on log-odds **for males** (reference group)
- β_3: How much the age effect **differs** for females compared to males

If $\beta_3 = 0$, the age effect is the same for both sexes (no interaction). Testing $H_0 : \beta_3 = 0$ tests whether

the interaction is significant.

6 From Probabilities to Classifications

Logistic regression outputs **probabilities** $\hat{p} = P(Y = 1|X)$. To make actual **classifications** (predict 0 or 1), we need a **threshold**.

6.1 The 0.5 Threshold

The most common approach:

$$\hat{Y} = \begin{cases} 1 & \text{if } \hat{p} \geq 0.5 \\ 0 & \text{if } \hat{p} < 0.5 \end{cases}$$

Intuition: If the probability of success is at least 50%, predict success.

Key Information

What about exactly 0.5?
In practice, this rarely matters. You can round up, round down, or flip a coin. The exact handling of 0.5 typically has negligible impact on overall performance.

6.2 Extension to Multiple Classes (K > 2)

When you have 3+ classes, the 0.5 rule doesn't work—you're not guaranteed any class has probability > 0.5.

Solution: Plurality Wins

Predict the class with the **highest probability**, even if it's below 0.5.

Example with 3 classes:

- $P(\text{CS}) = 0.35$
- $P(\text{Stat}) = 0.40$
- $P(\text{Other}) = 0.25$

Prediction: **Stat** (highest probability)

7 Decision Boundaries

A **decision boundary** is the line (or curve) where the model switches from predicting one class to another.

7.1 Mathematical Derivation

The decision boundary occurs where $P(Y = 1) = 0.5$. Let's trace through the math:

1. $P(Y = 1) = 0.5$
2. Odds $= \frac{0.5}{1-0.5} = \frac{0.5}{0.5} = 1$
3. Log-odds $= \ln(1) = 0$

Therefore, the decision boundary is defined by:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots = 0$$

Definition:

Decision Boundary The decision boundary in logistic regression is the set of all points where:

$$X\boldsymbol{\beta} = 0$$

Equivalently, it's where the log-odds equals zero, the odds equals one, and the probability equals 0.5.

7.2 Linear vs Non-Linear Decision Boundaries

Linear Decision Boundary:

If the model only includes linear terms $(X_1, X_2, \ldots)$, the equation $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$ defines a **straight line** (or hyperplane in higher dimensions).

Non-Linear Decision Boundary:

If the model includes:

- **Polynomial terms**: $X_1^2, X_2^2, X_1^3, \ldots$
- **Interaction terms**: $X_1 \cdot X_2$

Then the decision boundary becomes a **curve** (or curved surface).

Example:

Creating a Circular Decision Boundary To create a circular decision boundary, include quadratic terms:

$$\log(\text{Odds}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2$$

Setting this equal to zero can give a circle, ellipse, or other conic section depending on the coefficient values.

Warning

Complexity vs Overfitting

More complex decision boundaries (from polynomial features) can capture more intricate patterns, but they also risk **overfitting**—fitting the training data too closely and performing poorly on new data.

8 Regularization in Logistic Regression

When we have many features or polynomial terms, the model can become overfit. **Regularization** prevents this by penalizing large coefficients.

8.1 The Regularized Loss Function

Standard logistic regression minimizes binary cross-entropy. With **Ridge (L2) regularization**, we add a penalty:

$$\text{Loss}_{\text{regularized}} = \underbrace{-\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]}_{\text{Binary Cross-Entropy}} + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{\text{L2 Penalty}}$$

- λ controls the **strength of regularization**
- The penalty is applied to all β_j **except** β_0 (the intercept)

8.2 Effect of λ

- $\lambda = 0$: No regularization (standard logistic regression)
- λ small: Weak regularization, coefficients can be large
- λ large: Strong regularization, coefficients shrink toward zero
- $\lambda \to \infty$: All coefficients become zero (except intercept)

8.3 sklearn's C Parameter

Important:

Understanding C in sklearn sklearn uses `C` instead of λ, where $C \approx 1/\lambda$:

- **Large C** (e.g., 100): **Weak** regularization (small λ). Model fits training data more closely. Risk of overfitting.
- **Small C** (e.g., 0.01): **Strong** regularization (large λ). Coefficients shrink. Simpler model. Risk of underfitting.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Strong regularization (small C = large lambda)
model = LogisticRegression(penalty='l2', C=0.1)

# Find optimal C through cross-validation
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
grid_search = GridSearchCV(
    LogisticRegression(penalty='l2'),
    param_grid,
    cv=5,
```

```
    scoring='accuracy'
)
grid_search.fit(X_train, y_train)
print(f"Best C: {grid_search.best_params_['C']}")
```

Listing 2: Regularized Logistic Regression in sklearn

8.4 Visualizing Regularization's Effect on Decision Boundaries

Imagine an overfit model with a very "wiggly" decision boundary that perfectly separates training points. Regularization smooths out this boundary, making it more generalizable:

- **Unregularized**: Complex, possibly overfit boundary
- **Regularized**: Smoother, more generalizable boundary

9 Multiclass Classification

When the response variable has **more than two categories** (e.g., CS major, Stats major, Other), we need to extend binary logistic regression.

We focus on **nominal** categories (no inherent order). For **ordinal** categories (e.g., low/medium/high), specialized ordinal regression methods exist.

9.1 Approach 1: Multinomial Logistic Regression

Choose one class as the **reference group** (e.g., "Other"). Build $K - 1$ separate log-odds models comparing each class to the reference.

For 3 classes (CS, Stat, Other):

$$\log\left(\frac{P(\text{CS})}{P(\text{Other})}\right) = \beta_0^{(1)} + \beta_1^{(1)} X_1 + \cdots$$

$$\log\left(\frac{P(\text{Stat})}{P(\text{Other})}\right) = \beta_0^{(2)} + \beta_1^{(2)} X_1 + \cdots$$

Note: These are fit simultaneously, not independently, ensuring probabilities sum to 1.

```
from sklearn.linear_model import LogisticRegression

# Multinomial is now the default in newer sklearn versions
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
model.fit(X_train, y_train)

# Predict probabilities for all classes
probs = model.predict_proba(X_test)  # Shape: (n_samples, n_classes)
```

Listing 3: Multinomial Logistic Regression in sklearn

9.2 Approach 2: One-vs-Rest (OvR)

Build K separate binary classifiers:

- Classifier 1: CS vs Not-CS
- Classifier 2: Stat vs Not-Stat
- Classifier 3: Other vs Not-Other

For prediction, run all classifiers and pick the one with highest probability/confidence.

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression

model = OneVsRestClassifier(LogisticRegression())
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Listing 4: One-vs-Rest Classification

9.3 Softmax: Converting Scores to Probabilities

Both approaches produce "scores" for each class. To convert to proper probabilities (that sum to 1), we use the **Softmax function**:

$$P(Y = k|X) = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

where s_k is the score (log-odds) for class k.

Key Information

Properties of Softmax

- All outputs are between 0 and 1
- All outputs sum to exactly 1
- Higher scores get higher probabilities
- The relative differences are preserved (monotonic transformation)

10 Classification Evaluation: The Confusion Matrix

Once we have a classifier, how do we evaluate its performance? Simple accuracy can be misleading, especially with imbalanced classes.

10.1 The Confusion Matrix

The confusion matrix organizes all possible prediction outcomes:

		Predicted	
		Negative (0)	Positive (1)
Actual	Negative (0)	**TN** (True Negative)	**FP** (False Positive)
	Positive (1)	**FN** (False Negative)	**TP** (True Positive)

Definition:

Confusion Matrix Terms

- **True Positive (TP)**: Actually positive, predicted positive. Correct!
- **True Negative (TN)**: Actually negative, predicted negative. Correct!
- **False Positive (FP)**: Actually negative, predicted positive. Type I Error.
- **False Negative (FN)**: Actually positive, predicted negative. Type II Error.

Example:

Medical Diagnosis Context Consider a test for cancer:

- **TP**: Patient has cancer, test says positive. (Detected the disease)
- **TN**: Patient is healthy, test says negative. (Correctly cleared)
- **FP**: Patient is healthy, test says positive. (False alarm, unnecessary anxiety)
- **FN**: Patient has cancer, test says negative. (Missed diagnosis—**dangerous!**)

```python
from sklearn.metrics import confusion_matrix, classification_report

y_pred = model.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
# [[TN, FP],
#  [FN, TP]]

# Detailed report
print(classification_report(y_test, y_pred))
```

Listing 5: Computing Confusion Matrix in sklearn

11 Key Performance Metrics

Different metrics focus on different aspects of performance. The choice depends on the application and costs of different errors.

11.1 Sensitivity (Recall, True Positive Rate)

$$\text{Sensitivity} = \text{TPR} = \frac{TP}{TP + FN}$$

Question answered: Of all actually positive cases, what fraction did we correctly identify?

When it matters: When **missing a positive is costly**. In medical diagnosis, we don't want to miss cancer patients (minimize FN).

11.2 Specificity (True Negative Rate)

$$\text{Specificity} = \text{TNR} = \frac{TN}{TN + FP}$$

Question answered: Of all actually negative cases, what fraction did we correctly identify?

When it matters: When **false alarms are costly**. In spam filtering, we don't want to mark important emails as spam (minimize FP).

11.3 Precision (Positive Predictive Value)

$$\text{Precision} = \text{PPV} = \frac{TP}{TP + FP}$$

Question answered: Of all cases we predicted positive, what fraction are actually positive?

When it matters: When **acting on positive predictions is costly**. If treatment is expensive/harmful, we want predictions to be reliable.

11.4 False Positive Rate

$$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

Question answered: Of all actually negative cases, what fraction did we incorrectly classify as positive?

Warning

The Base Rate Problem (Bayes' Theorem)
Even with 99% sensitivity and 99% specificity, if a disease is very rare (e.g., 0.1% prevalence), most positive test results will be **false positives**!
This is why precision (PPV) can be low even with excellent sensitivity and specificity when the base rate is low.

12 The Threshold Trade-off

The classification threshold (default 0.5) is not sacred. Adjusting it creates a **trade-off** between sensitivity and specificity.

12.1 Lowering the Threshold (e.g., 0.5 → 0.3)

The model becomes more "eager" to predict positive:

- More True Positives (TP ↑) — **good**
- Fewer False Negatives (FN ↓) — **good**
- **Sensitivity increases**

But also:

- More False Positives (FP ↑) — **bad**
- Fewer True Negatives (TN ↓) — **bad**
- **Specificity decreases**

Use case: Medical screening where missing a case is unacceptable.

12.2 Raising the Threshold (e.g., 0.5 → 0.7)

The model becomes more "conservative" about predicting positive:

- Fewer False Positives (FP ↓) — **good**
- More True Negatives (TN ↑) — **good**
- **Specificity increases**

But also:

- Fewer True Positives (TP ↓) — **bad**
- More False Negatives (FN ↑) — **bad**
- **Sensitivity decreases**

Use case: When false positives are very costly (e.g., invasive surgery based on prediction).

Key Information

No Free Lunch

You cannot simultaneously maximize both sensitivity and specificity. Improving one typically hurts the other. The optimal threshold depends on the **relative costs** of false positives vs false negatives in your specific application.

13 ROC Curves and AUC

The **ROC Curve** (Receiver Operating Characteristic) visualizes the trade-off across **all possible thresholds**.

13.1 Constructing the ROC Curve

1. For each threshold from 0 to 1:
   - Calculate the True Positive Rate (TPR = Sensitivity)
   - Calculate the False Positive Rate (FPR = 1 - Specificity)
2. Plot each (FPR, TPR) pair as a point
3. Connect the points to form the curve

Axes:

- X-axis: False Positive Rate (FPR)
- Y-axis: True Positive Rate (TPR)

13.2 Interpreting the ROC Curve

- **Perfect classifier**: Goes from (0,0) to (0,1) to (1,1). It achieves 100% TPR with 0% FPR.
- **Random classifier**: The diagonal line from (0,0) to (1,1). TPR equals FPR at every threshold.
- **Good classifier**: Curves toward the upper-left corner (0,1).

13.3 AUC: Area Under the Curve

The **AUC** summarizes the entire ROC curve in a single number:

$$\text{AUC} = \int_0^1 \text{ROC}(x)\, dx$$

- **AUC = 1.0**: Perfect classifier
- **AUC = 0.5**: Random classifier (useless)
- **AUC = 0.8-0.9**: Good classifier
- **AUC < 0.5**: Worse than random (predictions are inverted)

Key Information

Probabilistic Interpretation of AUC
AUC equals the probability that a randomly chosen positive example is ranked higher (assigned higher probability) than a randomly chosen negative example.
AUC = 0.8 means: if you pick one positive and one negative case at random, there's an 80% chance the model assigns a higher probability to the positive case.

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

```

```python
# Get predicted probabilities
y_proba = model.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

# Calculate AUC
auc = roc_auc_score(y_test, y_proba)
print(f"AUC: {auc:.3f}")

# Plot
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Listing 6: ROC Curve and AUC in sklearn

14 Alternative Inference: Bootstrap and Permutation

Just as in linear regression, we have two approaches to inference:

14.1 Probabilistic Approach (Default)

Uses mathematical theory (Fisher's Information) to derive standard errors, confidence intervals, and p-values. This is what `statsmodels` provides.

Advantages:

- Fast computation
- Well-established theory

Disadvantages:

- Relies on asymptotic approximations (large sample)
- May not be accurate for small samples

14.2 Empirical Approach (Bootstrap/Permutation)

Bootstrap for Confidence Intervals:

1. Resample your data with replacement
2. Fit the model on each bootstrap sample
3. Calculate the coefficient
4. Repeat many times to get a distribution
5. Use percentiles for confidence intervals

Permutation for Hypothesis Testing:

1. Shuffle the Y values (break association with X)
2. Fit the model and record the coefficient
3. Repeat many times to get the null distribution
4. Compare observed coefficient to null distribution

Key Information

Advantages of Empirical Methods in Logistic Regression

In logistic regression, we don't have to worry about:

- Heteroscedasticity (variance depends on p, not separate)
- Normality of errors (no continuous errors)

The main assumption that matters is the correct specification of the model (right predictors, right functional form).

15 Practical Checklist

Before Your Midterm: Key Concepts Checklist

- ☐ Can you explain why logistic regression uses MLE instead of OLS?
- ☐ Do you understand why there's no closed-form solution (numerical methods needed)?
- ☐ Can you interpret β_0 for a model with a binary predictor (reference group concept)?
- ☐ Can you interpret β_1 as a difference in log-odds and e^{β_1} as an odds ratio?
- ☐ Do you understand the Z-distribution vs t-distribution distinction for inference?
- ☐ Can you write out how interaction terms create different slopes for different groups?
- ☐ Can you derive why $X\beta = 0$ is the decision boundary (from $P = 0.5$)?
- ☐ Do you understand how polynomial terms create non-linear decision boundaries?
- ☐ Can you explain regularization and the meaning of sklearn's C parameter?
- ☐ Do you know the difference between multinomial and one-vs-rest for multiclass?
- ☐ Can you calculate sensitivity, specificity, and precision from a confusion matrix?
- ☐ Do you understand the threshold trade-off (sensitivity vs specificity)?
- ☐ Can you interpret an ROC curve and explain what AUC measures?

Exam Tips from the Lecture

- For back-of-envelope calculations: use 2 for t-values and 1.96 for z-values
- Default answers if stuck: "cross-validation" or "it depends" or "MSE"
- Remember: logistic regression uses Z (not t) because variance is determined by p
- Coefficient interpretation: always talk about **odds**, not probability
- C in sklearn is **inverse** of λ: small C = strong regularization

16 Summary: Key Formulas

Logistic Regression Model

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

$$p = \frac{e^{X\beta}}{1+e^{X\beta}} = \frac{1}{1+e^{-X\beta}}$$

Loss Function

Binary Cross-Entropy:

$$\text{Loss} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

With L2 Regularization:

$$\text{Loss}_{\text{reg}} = \text{Loss} + \lambda \sum_{j=1}^{p} \beta_j^2$$

Evaluation Metrics

$$\text{Sensitivity (TPR)} = \frac{TP}{TP+FN}$$

$$\text{Specificity (TNR)} = \frac{TN}{TN+FP}$$

$$\text{Precision (PPV)} = \frac{TP}{TP+FP}$$

$$\text{FPR} = \frac{FP}{TN+FP} = 1 - \text{Specificity}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Key Relationships

- Probability $\rightarrow$ Odds: Odds $= \frac{p}{1-p}$
- Odds $\rightarrow$ Probability: $p = \frac{\text{Odds}}{1+\text{Odds}}$
- Odds $\rightarrow$ Log-Odds: log(Odds)
- Coefficient $\rightarrow$ Odds Ratio: e^{β}
- Decision Boundary: where $X\beta = 0$ (equivalently $p = 0.5$)
- sklearn's C: $C \approx 1/\lambda$ (inverse regularization strength)

Lecture 15: Multiclass Classification, ROC/AUC, and Introduction to Bayesian Inference

CS109A: Introduction to Data Science

Harvard University

■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 15
■ **Instructors:**	Pavlos Protopapas, Kevin Rader, Chris Tanner
■ **Topics:**	Multinomial Logistic Regression, One-vs-Rest, Softmax, Confusion Matrix, ROC Curves, AUC, Bayesian Inference, Beta Distribution, Beta-Binomial Model

Key Summary

This lecture extends classification to the multiclass setting and introduces tools for evaluating classifier performance. We then transition to Bayesian statistics, laying groundwork for Bayesian approaches to regression.

Part 1 - Multiclass Classification:

- Multinomial Logistic Regression: Using a reference class
- One-vs-Rest (OvR): Building K separate binary classifiers
- Softmax function: Converting scores to proper probabilities
- Making predictions when $K > 2$

Part 2 - Classification Evaluation:

- Confusion matrices: TP, FP, TN, FN
- Threshold selection and trade-offs
- ROC curves and AUC for model comparison

Part 3 - Bayesian Introduction:

- Bayes' Rule for parameter estimation
- The Beta distribution as a conjugate prior
- The Beta-Binomial model
- Preview of hierarchical models

Contents

1 Review: From Binary to Multiclass

In the previous lectures, we focused on **binary classification**: predicting whether $Y = 0$ or $Y = 1$. Logistic regression models the probability of success:

$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

But what if Y can take on **more than two values**?

Example:

Real-World Multiclass Problems

- **Student major prediction**: CS, Statistics, or Other
- **NFL play type**: Pass, Run, or Special Teams
- **Email classification**: Primary, Social, Promotions, or Spam
- **Medical diagnosis**: Healthy, Condition A, Condition B, Condition C

We need to extend logistic regression to handle $K > 2$ classes. There are two main approaches:

1. **Multinomial Logistic Regression**: Compare each class to a reference class
2. **One-vs-Rest (OvR)**: Build K separate binary classifiers

2 Multinomial Logistic Regression

2.1 The Setup

Suppose we have K classes labeled $0, 1, 2, \ldots, K-1$. Multinomial logistic regression:

1. Designates one class as the **reference group** (typically class $K-1$ or class 0)
2. Fits $K-1$ separate log-odds models comparing each other class to the reference

Example:

NFL Play Prediction We want to predict play type with $K = 3$ classes:

- Class 0: Special Teams (punt, field goal, etc.)
- Class 1: Pass play
- Class 2: Run play

Using **Class 0 as reference**, we fit two models:

Model 1 (Pass vs Special Teams):

$$\log\left(\frac{P(Y=1)}{P(Y=0)}\right) = \beta_0^{(1)} + \beta_1^{(1)} \cdot \text{Down} + \beta_2^{(1)} \cdot \text{Distance}$$

Model 2 (Run vs Special Teams):

$$\log\left(\frac{P(Y=2)}{P(Y=0)}\right) = \beta_0^{(2)} + \beta_1^{(2)} \cdot \text{Down} + \beta_2^{(2)} \cdot \text{Distance}$$

2.2 From Two Models to Three Probabilities

We have two equations but need three probabilities: $P(Y=0), P(Y=1), P(Y=2)$.

The key insight: **Probabilities must sum to 1**!

$$P(Y=0) + P(Y=1) + P(Y=2) = 1$$

With this third equation, we can solve for all three probabilities:

1. From Model 1: $P(Y=1) = P(Y=0) \cdot e^{\beta^{(1)}X}$
2. From Model 2: $P(Y=2) = P(Y=0) \cdot e^{\beta^{(2)}X}$
3. Sum constraint: $P(Y=0) + P(Y=0) \cdot e^{\beta^{(1)}X} + P(Y=0) \cdot e^{\beta^{(2)}X} = 1$

Solving for $P(Y=0)$:

$$P(Y=0) = \frac{1}{1 + e^{\beta^{(1)}X} + e^{\beta^{(2)}X}}$$

And then:

$$P(Y=k) = \frac{e^{\beta^{(k)}X}}{1 + e^{\beta^{(1)}X} + e^{\beta^{(2)}X}} \quad \text{for } k \in \{1, 2\}$$

2.3 Interpreting the Output

Example:

Reading sklearn's Multinomial Output sklearn reports coefficients for **all K classes** (not just K-1):

```python
model = LogisticRegression(multi_class='multinomial')
model.fit(X, y)
print(model.coef_)  # Shape: (3, 2) for 3 classes, 2 features
```

Output might look like:

```
[[-6.22,  1.67,  0.10],   # Class 0 coefficients
 [ 1.86, -0.04, -0.02],   # Class 1 coefficients
 [-1.64, -0.63, -0.08]]   # Class 2 coefficients
```

Why 3 sets of coefficients when theory says K-1?
sklearn internally **renormalizes** the coefficients so that each can be interpreted as "Class k vs Not Class k" rather than "Class k vs Reference." This makes predictions easier to compute but changes the interpretation slightly.
For the first class (Special Teams):

$$\log\left(\frac{P(Y=0)}{P(Y\neq 0)}\right) = -6.22 + 1.67 \cdot \text{Down} + 0.10 \cdot \text{Distance}$$

Interpretation: As "Down" increases (approaching 4th down), the probability of a special teams play increases. As "Distance" increases, special teams also becomes more likely (punting on 4th and long).

3 One-vs-Rest (OvR) Classification

3.1 The Approach

One-vs-Rest takes a different strategy: instead of comparing to a reference group, we build **K completely separate binary classifiers**, one for each class.

- **Classifier 1**: Class 0 vs (Classes 1, 2, ..., K-1)
- **Classifier 2**: Class 1 vs (Classes 0, 2, ..., K-1)
- ⋮
- **Classifier K**: Class K-1 vs (Classes 0, 1, ..., K-2)

Example:

OvR for NFL Plays **Classifier 1 (Special Teams vs Everything Else):**

- Positive class: Special Teams (Class 0)
- Negative class: Pass + Run (Classes 1, 2 combined)

Classifier 2 (Pass vs Everything Else):

- Positive class: Pass (Class 1)
- Negative class: Special Teams + Run (Classes 0, 2 combined)

Classifier 3 (Run vs Everything Else):

- Positive class: Run (Class 2)
- Negative class: Special Teams + Pass (Classes 0, 1 combined)

3.2 The Problem: Probabilities Don't Sum to 1

Each classifier outputs its own probability:

- $p_0 = P(\text{Special Teams vs Others})$
- $p_1 = P(\text{Pass vs Others})$
- $p_2 = P(\text{Run vs Others})$

But these three probabilities typically **do not sum to 1**! They come from independent models, each trained on slightly different data configurations.

3.3 Solution: The Softmax Function

To convert these scores into proper probabilities, we use the **Softmax function**:

Definition:

Softmax Function Given scores $(s_1, s_2, \ldots, s_K)$ for K classes, the softmax function converts them to probabilities:

$$P(Y = k) = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

Properties:

- All outputs are positive (due to exponential)
- All outputs are between 0 and 1
- All outputs **sum to exactly 1**
- Larger scores get larger probabilities (monotonic)

Example:

Softmax in Action Suppose our three classifiers output log-odds (scores):

- $s_0 = -2$ (Special Teams)
- $s_1 = 1.5$ (Pass)
- $s_2 = 0.8$ (Run)

Apply softmax:

$$\sum e^{s_j} = e^{-2} + e^{1.5} + e^{0.8} = 0.135 + 4.48 + 2.23 = 6.84$$

$$P(Y = 0) = \frac{0.135}{6.84} = 0.02$$
$$P(Y = 1) = \frac{4.48}{6.84} = 0.66$$
$$P(Y = 2) = \frac{2.23}{6.84} = 0.33$$

Now the probabilities sum to 1, and we'd predict **Pass** (Class 1).

3.4 Multinomial vs OvR: Which to Use?

Aspect	**Multinomial**	**OvR**
Number of models	$K - 1$	K
Training	Joint optimization	Independent classifiers
Data usage	All data, structured	May discard some structure
Decision boundaries	One per pair (vs reference)	One per class (vs all)
Performance	Often similar	Often similar

Key Information

How to Choose?

In practice, both methods often give very similar results. The best approach:

1. Try both methods
2. Compare using **cross-validation**
3. Choose based on test set performance (not training performance!)

4 Making Predictions with K > 2 Classes

4.1 From Probabilities to Classifications

In binary classification, we used the threshold $P(Y = 1) > 0.5$ to predict class 1.

With $K > 2$ classes, no single class is guaranteed to have probability > 0.5. Instead, we use the **plurality rule**:

$$\hat{Y} = \arg\max_k P(Y = k|X)$$

Simply predict the class with the highest probability, even if that probability is below 0.5.

Example:

Plurality Prediction Suppose our model predicts:

- $P(\text{Special Teams}) = 0.05$
- $P(\text{Pass}) = 0.55$
- $P(\text{Run}) = 0.40$

Prediction: Pass (highest probability)

Another example:

- $P(\text{Special Teams}) = 0.10$
- $P(\text{Pass}) = 0.45$
- $P(\text{Run}) = 0.45$

Prediction: Either Pass or Run (tie—implementation dependent)

4.2 Class Imbalance Problems

Warning

When Classification Always Predicts the Same Class

If one class dominates the data (e.g., 66% of NFL plays are passes), the model might predict "Pass" for almost every observation—and still achieve 66% accuracy!

The cocaine example from lecture: If you're predicting whether someone is currently high on cocaine, you'd predict "No" for everyone and be right 99.9%+ of the time.

Key insight: The model might still capture meaningful relationships through the **probabilities**, even if the pure **classifications** are all the same. Always examine predicted probabilities, not just classifications.

4.3 Loss Function for Multiclass

Binary classification uses **Binary Cross-Entropy**:

$$\text{Loss} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Multiclass classification generalizes this to **Cross-Entropy** (or Multinomial Logistic Loss):

$$\text{Loss} = -\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbf{1}_{[y_i=k]} \log(p_{ik})$$

where p_{ik} is the predicted probability that observation i belongs to class k.

Regularization (L1/Lasso or L2/Ridge) can still be applied to this loss function to prevent overfitting.

5 Review: The Confusion Matrix

The **confusion matrix** is the foundation of classification evaluation. For binary classification:

		Predicted	
		Negative (0)	Positive (1)
Actual	Negative (0)	TN	FP
	Positive (1)	FN	TP

Definition:

Confusion Matrix Elements

- **True Positive (TP)**: Actually positive, predicted positive. Correct!
- **True Negative (TN)**: Actually negative, predicted negative. Correct!
- **False Positive (FP)**: Actually negative, predicted positive. Type I Error.
- **False Negative (FN)**: Actually positive, predicted negative. Type II Error.

Example:

Heart Disease Example From the lecture, predicting heart disease using age, sex, and their interaction:

	Predicted: No	Predicted: Yes
Actual: No	110 (TN)	54 (FP)
Actual: Yes	53 (FN)	86 (TP)

Is this a useful model?

- Among actual negatives (110 + 54 = 164): 110/164 = 67% correctly identified
- Among actual positives (53 + 86 = 139): 86/139 = 62% correctly identified
- Better than random chance (which would be 50/50)
- Not perfect, but has discriminatory power

6 The Threshold Trade-off

Logistic regression outputs **probabilities**, not classifications. We convert to classifications using a **threshold**:

$$\hat{Y} = \begin{cases} 1 & \text{if } \hat{p} \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

The default threshold is 0.5, but this can be changed!

6.1 Effect of Changing the Threshold

Lowering the threshold (e.g., 0.5 → 0.4):

- Model predicts "positive" more easily
- **TP increases** (good), **FN decreases** (good)
- **FP increases** (bad), **TN decreases** (bad)
- **Use case**: Medical screening—we don't want to miss sick patients (minimize FN)

Raising the threshold (e.g., 0.5 → 0.6):

- Model predicts "positive" more conservatively
- **FP decreases** (good), **TN increases** (good)
- **TP decreases** (bad), **FN increases** (bad)
- **Use case**: Spam filtering—we don't want to lose important emails (minimize FP)

Important:

The Fundamental Trade-off **You cannot simultaneously minimize both FP and FN.** Reducing one type of error inevitably increases the other. The optimal threshold depends on the **relative costs** of each type of error in your specific application.

7 ROC Curves

The **ROC Curve** (Receiver Operating Characteristic) visualizes classifier performance across **all possible thresholds**.

7.1 Key Metrics

Definition:

TPR and FPR **True Positive Rate (TPR)** = Sensitivity = Recall:

$$\text{TPR} = \frac{TP}{TP + FN} = P(\hat{Y} = 1 | Y = 1)$$

"Of all actual positives, what fraction did we catch?"
False Positive Rate (FPR) = 1 - Specificity:

$$\text{FPR} = \frac{FP}{FP + TN} = P(\hat{Y} = 1 | Y = 0)$$

"Of all actual negatives, what fraction did we falsely classify as positive?"

7.2 Constructing the ROC Curve

1. For each possible threshold (from 0 to 1):
   - Classify all observations using that threshold
   - Calculate the resulting TPR and FPR
   - Plot the point (FPR, TPR)
2. Connect all points to form the curve

7.3 Interpreting the ROC Curve

Key Reference Points:

- **(0, 0)**: Threshold = 1 (predict everyone negative)
- **(1, 1)**: Threshold = 0 (predict everyone positive)
- **(0, 1)**: Perfect classifier (100% TPR, 0% FPR)

Key Reference Lines:

- **Diagonal (y = x)**: Random classifier. A coin flip with probability p gives point (p, p).
- **Upper-left corner**: Ideal. We want the curve to hug this corner.

Example:

Reading ROC Curves If you see three curves:

- **Blue curve**: Hugs upper-left corner tightly
- **Green curve**: Moderately above the diagonal

- **Red dashed line**: The diagonal (random baseline)

The blue model is best—for any given FPR, it achieves higher TPR than the others.

8 AUC: Area Under the Curve

Comparing ROC curves visually can be difficult, especially when curves cross. The **AUC** (Area Under the Curve) summarizes performance in a single number.

8.1 Computing AUC

AUC is literally the area under the ROC curve:

$$\text{AUC} = \int_0^1 \text{ROC}(x)\, dx$$

In practice, we approximate this using the trapezoidal rule over the discrete threshold points.

8.2 Interpreting AUC

- **AUC = 1.0**: Perfect classifier
- **AUC = 0.5**: Random classifier (no better than coin flip)
- **AUC < 0.5**: Worse than random (predictions are inverted)

Key Information

Probabilistic Interpretation of AUC

AUC equals the probability that a randomly chosen positive example is ranked higher (assigned higher predicted probability) than a randomly chosen negative example.

AUC = 0.8 means: If you pick one positive and one negative case at random, there's an 80% chance the model assigns higher probability to the positive case.

```python
from sklearn.metrics import roc_curve, roc_auc_score

# Get predicted probabilities
y_proba = model.predict_proba(X_test)[:, 1]

# Calculate ROC curve points
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

# Calculate AUC
auc_score = roc_auc_score(y_test, y_proba)
print(f"AUC: {auc_score:.3f}")
```

Listing 1: Computing ROC and AUC in sklearn

9 Introduction to Bayesian Inference

So far, we've used the **frequentist** approach:

- Parameters (β) are fixed but unknown constants
- We estimate them using data (MLE, OLS)
- Uncertainty is expressed through confidence intervals

The **Bayesian** approach takes a fundamentally different view:

Definition:

Bayesian Philosophy In Bayesian statistics, parameters are **random variables** with probability distributions.
We start with a **prior belief** about the parameter, observe data, and update our belief to obtain a **posterior distribution**.

9.1 Bayes' Rule for Parameter Estimation

$$\underbrace{f(\theta|X)}_{\text{Posterior}} = \frac{\overbrace{f(X|\theta)}^{\text{Likelihood}} \cdot \overbrace{f(\theta)}^{\text{Prior}}}{\underbrace{f(X)}_{\text{Normalizing constant}}}$$

In practice, we often ignore the normalizing constant and write:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

The components:

- **Prior** $f(\theta)$: Our belief about θ *before* seeing the data
- **Likelihood** $f(X|\theta)$: Probability of observing data X if θ is the true value
- **Posterior** $f(\theta|X)$: Our updated belief about θ *after* seeing the data

Key Summary

Bayesian Inference in One Sentence:
Prior belief $\times$ Evidence from data $\rightarrow$ Updated belief

10 The Beta Distribution

Before we can do Bayesian inference for classification, we need a distribution for modeling probabilities. Enter the **Beta distribution**.

10.1 Why Beta?

We need a distribution that:

1. Is defined on $[0, 1]$ (since probabilities are between 0 and 1)
2. Is flexible enough to represent different prior beliefs
3. Works nicely with Bernoulli/Binomial likelihoods

The Beta distribution satisfies all three!

10.2 The Beta Distribution

Definition:

Beta Distribution A random variable $X \sim \text{Beta}(\alpha, \beta)$ has PDF:

$$f(x|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad x \in [0, 1]$$

where $\Gamma(\cdot)$ is the gamma function (generalization of factorial: $\Gamma(n) = (n-1)!$ for integers).

Key properties:

- Mean: $E[X] = \frac{\alpha}{\alpha+\beta}$
- Mode: $\frac{\alpha-1}{\alpha+\beta-2}$ (for $\alpha, \beta > 1$)
- Variance: $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

10.3 Shape of Beta Distributions

- **Beta**$(1, 1)$: Uniform distribution on $[0, 1]$. "I have no prior information."
- **Beta**$(10, 10)$: Symmetric, peaked at 0.5. "I believe $p \approx 0.5$."
- **Beta**$(2, 5)$: Skewed left, mean ≈ 0.29. "I believe p is small."
- **Beta**$(5, 2)$: Skewed right, mean ≈ 0.71. "I believe p is large."
- **Beta**$(0.5, 0.5)$: U-shaped. "I believe p is either very small or very large."

Key Information

Intuitive Interpretation

Think of $\alpha - 1$ as "prior successes" and $\beta - 1$ as "prior failures" you've imagined before seeing real data.

Beta$(1, 1)$ = 0 prior successes, 0 prior failures (no information)

Beta$(10, 10)$ = 9 prior successes, 9 prior failures (believe coin is fair)

11 The Beta-Binomial Model

Now we put it all together: using the Beta distribution as a prior for a Binomial likelihood.

11.1 The Setup

Goal: Estimate the probability p of success (e.g., probability a coin lands heads).

Data: n independent trials with k successes and $n - k$ failures.

Model:

- **Likelihood**: $X|p \sim \text{Binomial}(n, p)$
- **Prior**: $p \sim \text{Beta}(\alpha_0, \beta_0)$

11.2 Computing the Posterior

Using Bayes' rule:

$$f(p|X) \propto f(X|p) \cdot f(p)$$

Likelihood (ignoring constants not involving p):

$$f(X|p) \propto p^k(1-p)^{n-k}$$

Prior:

$$f(p) \propto p^{\alpha_0-1}(1-p)^{\beta_0-1}$$

Posterior:

$$\begin{aligned} f(p|X) &\propto p^k(1-p)^{n-k} \cdot p^{\alpha_0-1}(1-p)^{\beta_0-1} \\ &= p^{(\alpha_0+k)-1}(1-p)^{(\beta_0+n-k)-1} \end{aligned}$$

This is a **Beta distribution**!

Important:

Beta-Binomial Update Rule

$$\begin{aligned} \text{Prior}: &\quad p \sim \text{Beta}(\alpha_0, \beta_0) \\ \text{Data}: &\quad k \text{ successes}, n-k \text{ failures} \\ \text{Posterior}: &\quad p|X \sim \text{Beta}(\alpha_0 + k, \beta_0 + n - k) \end{aligned}$$

The posterior is just the prior with successes and failures added!

11.3 Conjugate Priors

When the prior and posterior belong to the **same family of distributions**, the prior is called a **conjugate prior** for that likelihood.

The Beta distribution is the conjugate prior for the Bernoulli/Binomial likelihood. This is mathematically convenient—the posterior has a known, closed-form distribution.

Example:

Coin Flipping **Prior**: I have no strong belief, so I use Beta$(1, 1)$ (uniform).

$$E[p] = \frac{1}{2} = 0.5$$

Data: I flip the coin 10 times and get 7 heads, 3 tails.
Posterior: Beta$(1 + 7, 1 + 3) =$ Beta$(8, 4)$

$$E[p|\text{data}] = \frac{8}{12} = 0.67$$

My belief shifted from 0.5 to 0.67 based on the evidence!

12 Preview: Hierarchical Models

What if we have **grouped data**? For example, estimating shooting percentages for multiple NBA players?

12.1 The Problem

Option 1 (Pooled): Assume all players have the same p. Too restrictive—LeBron James is different from a rookie.

Option 2 (Separate): Estimate each player's p independently. Problem: A rookie with 5 shots has very uncertain estimate.

Option 3 (Hierarchical): The best of both worlds!

12.2 The Hierarchical Approach

1. **Level 1 (Data)**: Each player j's shots follow their own probability p_j
2. **Level 2 (Players)**: The p_j's themselves come from a common distribution

$$p_j \sim \text{Beta}(\alpha, \beta)$$

3. **Level 3 (League)**: The hyperparameters α, β might have their own prior (hyperprior)

Key Information

Why Hierarchical Models Work

Hierarchical models **share information** across groups.

- LeBron James (1000 shots): His p_j is mostly determined by his own data
- Rookie (10 shots): His p_j is **shrunk toward the league average**, borrowing strength from other players

This "shrinkage" produces better estimates for players with limited data!

Warning

Bayesian Logistic Regression: Not Beta!

Can we use Beta as a prior for logistic regression? **No!**

- Beta is for probabilities $p \in [0, 1]$
- Logistic regression parameters β can be any real number $(-\infty, \infty)$

For logistic regression, we typically use **Normal distributions** as priors for β:

$$\beta_j \sim N(0, \sigma^2)$$

A prior centered at 0 means "I expect this coefficient to be small"—similar to Ridge regularization!

13 Summary

Multiclass Classification

- **Multinomial**: $K-1$ models comparing each class to a reference
- **OvR**: K separate "class vs others" classifiers
- **Softmax**: Converts scores to probabilities summing to 1
- **Prediction**: Choose class with highest probability (plurality)

Classification Evaluation

- **Confusion Matrix**: TP, FP, TN, FN
- **TPR (Sensitivity)**: $\frac{TP}{TP+FN}$ — catching positives
- **FPR**: $\frac{FP}{FP+TN}$ — false alarms
- **ROC Curve**: TPR vs FPR across all thresholds
- **AUC**: Area under ROC; 1 = perfect, 0.5 = random

Bayesian Inference

- **Bayes' Rule**: Posterior $\propto$ Likelihood $\times$ Prior
- **Beta Distribution**: Prior for probabilities, $p \in [0,1]$
- **Beta-Binomial**: $\text{Beta}(\alpha,\beta) \to \text{Beta}(\alpha+k, \beta+n-k)$
- **Conjugate Prior**: Prior and posterior same family
- **Hierarchical Models**: Share information across groups

Key Formulas

Softmax: $P(Y=k) = \frac{e^{s_k}}{\sum_j e^{s_j}}$
TPR: $\frac{TP}{TP+FN}$ **FPR:** $\frac{FP}{FP+TN}$
Beta Mean: $E[X] = \frac{\alpha}{\alpha+\beta}$
Beta-Binomial Update: $\text{Beta}(\alpha_0+k, \beta_0+n-k)$

Lecture 16: Hierarchical Models and Bayesian Logistic Regression

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 16
- **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Tanner
- **Topics:** Logistic Regression Review, Log-Scale Regression, Beta-Binomial Model, Hierarchical Models, Bayesian Logistic Regression, Posterior Predictive Distribution

Key Summary

This lecture applies Bayesian thinking to logistic regression and introduces hierarchical models—one of the most powerful tools for handling structured data with groups.

Key Topics:

- Review: Interpreting logistic regression coefficients (NBA shooting example)
- Log-log regression for multiplicative relationships (housing prices)
- Review of the Beta-Binomial model and conjugacy
- Hierarchical models: Why and how to model grouped data
- The shrinkage effect: Borrowing strength across groups
- Bayesian logistic regression: Priors on β coefficients
- Posterior predictive distributions for forecasting
- Preview: When conjugacy fails, we need simulation (MCMC)

Contents

1 Review: Interpreting Logistic Regression

Before diving into new material, let's solidify our ability to interpret logistic regression models through a concrete example.

1.1 NBA Shooting Example

Data: All NBA field goal attempts from the previous season.

Response: $Y = 1$ if shot is successful, $Y = 0$ if missed.

Predictor: Distance from the hoop (in feet).

After fitting a logistic regression model:

$$\log\left(\frac{p}{1-p}\right) = 0.796 - 0.0474 \times \text{Distance}$$

1.2 Step-by-Step Interpretation

1. Interpreting the Intercept (0.796):

The intercept represents the log-odds when Distance = 0 (a layup right at the basket).

- Log-odds = 0.796
- Odds = $e^{0.796} = 2.22$
- Probability = $\frac{2.22}{1+2.22} = \frac{e^{0.796}}{1+e^{0.796}} \approx 0.69$

Interpretation: A shot from 0 feet has approximately a 69% probability of success.

2. Interpreting the Slope (-0.0474):

The slope represents the change in log-odds for each additional foot of distance.

- For each 1-foot increase in distance, log-odds **decreases** by 0.0474
- Odds ratio = $e^{-0.0474} \approx 0.954$
- Each additional foot **multiplies** the odds by 0.954 (a 4.6% decrease)

Interpretation: Longer shots are harder—makes sense!

3. Finding the Classification Boundary:

Where does the predicted probability equal 0.5?

At $P = 0.5$: Log-odds = 0

$$0 = 0.796 - 0.0474 \times \text{Distance}$$
$$\text{Distance} = \frac{0.796}{0.0474} \approx 16.8 \text{ feet}$$

Interpretation: Shots from less than 17 feet are predicted as makes; shots from beyond 17 feet are predicted as misses.

Key Information

The Complete Interpretation Recipe

1. Write out the model equation
2. Interpret intercept: Log-odds when all predictors = 0
3. Convert to probability: $p = \frac{e^{\text{log-odds}}}{1+e^{\text{log-odds}}}$
4. Interpret slopes: Change in log-odds per unit change
5. Convert to odds ratio: e^{β} = multiplicative change in odds
6. Find decision boundary: Set log-odds = 0, solve for X

2 Log-Log Regression: Multiplicative Models

Sometimes the relationship between X and Y is **multiplicative** rather than additive. This is common in financial and economic data.

2.1 The Problem with Standard Linear Regression

Consider predicting house prices from square footage. Both variables:

- Are strictly positive
- Have right-skewed distributions
- Show heteroscedasticity (variance increases with the mean)

2.2 The Solution: Log-Transform Both Variables

If we take the logarithm of both X and Y:

$$\log_2(Y) = \beta_0 + \beta_1 \log_2(X)$$

Example:

Housing Price Example Model on log-log scale:

$$\log_2(\text{Price}) = 12.46 + 0.722 \times \log_2(\text{SqFt})$$

Interpreting the slope (0.722):
What does a 1-unit change in $\log_2(\text{SqFt})$ mean?

- A 1-unit increase in $\log_2(X)$ means X **doubles**
- This produces a 0.722-unit increase in $\log_2(Y)$
- A 0.722 increase in $\log_2(Y)$ means Y is multiplied by $2^{0.722} \approx 1.65$

Final interpretation: When you **double** the square footage, the price increases by approximately **65%**.

Warning

Why Base 2?
Using $\log_2$ makes interpretation intuitive:

- 1 unit change = **doubling**
- Easy to conceptualize

If you use ln (natural log), a 1-unit change means multiplying by $e \approx 2.718$, which is harder to interpret.

Alternative interpretation (for natural log): For small β_1, approximately $\beta_1 \times 100\%$ change in Y per 1% change in X.

3 Review: Beta-Binomial Model

The Beta-Binomial model is the foundation for understanding Bayesian approaches to classification.

3.1 The Setup

Data: n independent trials, $\sum y_i$ successes, $n - \sum y_i$ failures.

Likelihood: Bernoulli (or Binomial)

$$Y_i|p \sim \text{Bernoulli}(p)$$

Prior: Beta distribution on p

$$p \sim \text{Beta}(a_0, b_0)$$

Hyperparameters: a_0 and b_0 encode our prior belief about p.

3.2 Conjugacy: Why Beta is Special

When we multiply the prior and likelihood:

$$\begin{aligned}\text{Posterior} &\propto \text{Likelihood} \times \text{Prior} \\ f(p|Y) &\propto p^{\sum y_i}(1-p)^{n-\sum y_i} \times p^{a_0-1}(1-p)^{b_0-1} \\ &= p^{(a_0+\sum y_i)-1}(1-p)^{(b_0+n-\sum y_i)-1}\end{aligned}$$

This is exactly a Beta distribution!

Important:

Beta-Binomial Conjugacy

$$\text{Prior: } p \sim \text{Beta}(a_0, b_0)$$

$$\text{Posterior: } p|Y \sim \text{Beta}(a_0 + \sum y_i, b_0 + n - \sum y_i)$$

The posterior is just the prior with successes and failures "added in."

3.3 Posterior Mean: A Weighted Average

The posterior mean is:

$$E[p|Y] = \frac{a_0 + \sum y_i}{a_0 + b_0 + n}$$

This is a **weighted average** of:

- The prior mean: $\frac{a_0}{a_0+b_0}$
- The MLE (sample proportion): $\frac{\sum y_i}{n}$

Key insight: As n (data) increases, the posterior mean approaches the MLE. The prior matters less

with more data.

4 Why Hierarchical Models?

Hierarchical models address a fundamental problem: data often has **natural grouping structure**.

4.1 Examples of Hierarchically Structured Data

- **Government**: Individual voters within counties within states within regions
- **Education**: Students within classrooms within schools within districts
- **Medicine**: Repeated measurements within patients within hospitals
- **Biology**: Cells within tissues within organs within organisms
- **Sports**: Shots within players within teams within leagues

4.2 The NBA Shooting Problem

Goal: Predict shot success based on distance, accounting for player ability.

Data structure:

- Response: Y_{ij} = success/failure of shot i by player j
- Predictor: X_{ij} = distance of shot i by player j
- Grouping: 600 different players, varying number of shots each

4.3 Three Approaches

Approach 1: Complete Pooling (Ignore Players)

$$\log\left(\frac{p_{ij}}{1-p_{ij}}\right) = \beta_0 + \beta_1 \times \text{Distance}_{ij}$$

Problem: Assumes all players are identical. Ignores obvious differences (LeBron vs a rookie).

Approach 2: No Pooling (One-Hot Encoding)

$$\log\left(\frac{p_{ij}}{1-p_{ij}}\right) = \beta_0 + \beta_1 \times \text{Distance}_{ij} + \sum_j \gamma_j \times \mathbf{1}[\text{Player} = j]$$

Problem: 600+ parameters! Severe overfitting, especially for players with few shots.

Approach 3: Hierarchical Model (Partial Pooling)

The best of both worlds!

5 Hierarchical Models: The Setup

5.1 The Model

Level 1 (Shots within Players):

$$\log\left(\frac{p_{ij}}{1-p_{ij}}\right) = \alpha_j + \beta_1 \times \text{Distance}_{ij}$$

Each player j has their own intercept α_j (baseline shooting ability).

Level 2 (Players within League):

$$\alpha_j \sim N(\alpha_0, \sigma_\alpha^2)$$

The player intercepts are drawn from a common distribution:

- α_0 = league-average baseline ability
- σ_α^2 = variance of abilities across players

Definition:

Hierarchical Model In a hierarchical model, **parameters** at one level are treated as **random variables** drawn from a distribution defined at a higher level.
This creates "partial pooling"—each group's estimate is informed by:

1. Its own data
2. The overall distribution of all groups

5.2 What Are We Estimating?

Fixed effects (same for everyone):

- β_1: Effect of distance on log-odds (assumed same for all players)

Hyperparameters (describe the population):

- α_0: Mean player ability
- σ_α^2: Variance of player abilities

Random effects (vary by group):

- α_j for each player j: Individual player abilities

6 The Shrinkage Effect

The magic of hierarchical models lies in **shrinkage**—pulling extreme estimates toward the group mean.

6.1 The Problem with OLS

Consider two players:

- **Alandis Williams**: 1 shot, 1 made (100% in sample)
- **Mac McClung**: 2 shots, 0 made (0% in sample)

With standard OLS (no pooling):

- Williams gets coefficient $\to +\infty$ (predicted 100% from any distance!)
- McClung gets coefficient $\to -\infty$ (predicted 0% from any distance!)

These are terrible estimates based on almost no data.

6.2 The Hierarchical Solution

With hierarchical modeling:

- Players with many shots: Estimates based mostly on their own data
- Players with few shots: Estimates **shrunk** toward the league average

Example:

Shrinkage in Action **Top 5 players (OLS estimates):**

- Alandis Williams (never heard of): Coefficient $\approx +10$ (100% predicted!)

n very few shots

Top 5 players (Hierarchical estimates):

- Jarrett Allen, SGA, Damian Lillard... (famous, high-volume shooters!)

The hierarchical model **automatically discounts** estimates based on small samples and gives appropriate credit to players with substantial evidence.

Key Information

How Does Sample Size Enter the Model?

The α_j distribution connects each player's shots to the overall population. A player with:

- **Many shots**: Strong evidence from their Bernoulli trials pulls α_j toward their sample mean
- **Few shots**: Weak evidence, so α_j is dominated by the prior (population mean α_0)

The math automatically handles this trade-off!

7 Extending Hierarchical Models

7.1 Random Slopes

In our current model, β_1 (distance effect) is the same for all players. But maybe:

- Steph Curry is **better** at long-range shots
- Other players are **worse** from distance

We can let the slope vary too:

$$\log\left(\frac{p_{ij}}{1-p_{ij}}\right) = \alpha_j + \beta_{1j} \times \text{Distance}_{ij}$$

where both α_j and β_{1j} are drawn from distributions.

7.2 Hyperpriors: Priors on Hyperparameters

What if we're uncertain about α_0 and σ_α^2?

We can add another layer:

- $\alpha_0 \sim N(\mu_0, \tau^2)$ (hyperprior on mean)
- $\sigma_\alpha^2 \sim$ Inverse-Gamma$(\ldots)$ (hyperprior on variance)

It's turtles all the way down!

In practice, we usually stop at 2-3 levels.

7.3 Fully Bayesian vs Empirical Bayes

Fully Bayesian: Put priors on ALL unknown parameters $(\alpha_0, \sigma_\alpha^2, \beta_1)$

Empirical Bayes: Estimate hyperparameters from the data (like treating $\alpha_0, \sigma_\alpha^2$ as fixed unknowns)

The hierarchical model we discussed is often fit in an "empirical Bayes" spirit, which is a blend of frequentist and Bayesian thinking.

8 Bayesian Logistic Regression

Now let's apply full Bayesian thinking to logistic regression parameters.

8.1 Why Not Use Beta Priors?

In the Beta-Binomial model, we put a Beta prior on p.

Can we do the same for logistic regression?

NO! Here's why:

- The Beta distribution has support $[0, 1]$
- In logistic regression, our parameters are $\beta_0, \beta_1, \ldots$ (the coefficients)
- These coefficients are on the **log-odds scale**, which is $(-\infty, +\infty)$
- A Beta prior would be inappropriate!

8.2 Normal Priors on β

Instead, we use **Normal priors**:

$$\beta_j \sim N(\mu_j, \sigma_j^2)$$

Common choices:

- $\mu_j = 0$: We expect coefficients to be near zero (conservative)
- σ_j^2 controls how strongly we believe this

Key Information

Connection to Ridge Regression

A Normal prior centered at 0 is the Bayesian interpretation of Ridge (L2) regularization!

- Strong prior (σ^2 small) = Strong regularization
- Weak prior (σ^2 large) = Weak regularization

8.3 The Loss of Conjugacy

Problem: Normal prior $\times$ Bernoulli likelihood $\neq$ Nice closed form!

The posterior distribution doesn't have a recognizable form. We can't write down:

$$E[\beta_j|\text{data}] = \text{simple formula}$$

Solution: Simulation! (MCMC, covered in next lecture)

9 Posterior Predictive Distribution

Once we have a model, we want to predict future observations.

9.1 Frequentist Prediction

In standard regression, prediction is simple:

$$\hat{y}_{\text{new}} = \hat{\beta}_0 + \hat{\beta}_1 x_{\text{new}}$$

Plug in point estimates and get a point prediction.

9.2 Bayesian Prediction

In Bayesian inference, parameters are random variables! We can't just "plug in" single values.

The posterior predictive distribution accounts for:

1. Uncertainty in the parameters (posterior distribution)
2. Inherent randomness in the outcome (likelihood)

Definition:

Posterior Predictive Distribution

$$f(\tilde{y}|\text{data}) = \int f(\tilde{y}|\theta) f(\theta|\text{data}) d\theta$$

This marginalizes out the uncertainty in θ by integrating over all possible parameter values, weighted by their posterior probability.

9.3 Intuition

To predict a new shot for an NBA player:

1. Draw a value of α_j, β_1 from their posterior distribution
2. Use those values to compute p for the new shot
3. Draw a success/failure from Bernoulli(p)
4. Repeat many times to get the full distribution of predictions

This gives us not just a point prediction, but a **distribution** capturing all our uncertainty.

10 Preview: When We Can't Solve Analytically

10.1 The Challenge

For many realistic Bayesian models:

- No conjugacy (posterior doesn't have nice form)
- High-dimensional parameter space
- Complex hierarchical structure

We cannot:

- Write down $f(\theta|\text{data})$ in closed form
- Compute $E[\theta|\text{data}]$ analytically
- Integrate to get posterior predictive distributions

10.2 The Solution: MCMC

Markov Chain Monte Carlo (MCMC) is a family of algorithms that:

1. Generate samples from the posterior distribution
2. Without needing to know its exact form
3. Use those samples to estimate quantities of interest

Key insight: If we can evaluate the posterior *up to a constant* (i.e., we know likelihood $\times$ prior), we can still sample from it!

10.3 Coming Up Next

In the next lecture, we'll cover:

- Monte Carlo integration
- The Metropolis-Hastings algorithm
- Practical considerations for MCMC

11 Summary

Logistic Regression Interpretation

1. Write out: $\log(p/(1-p)) = \beta_0 + \beta_1 X$
2. Intercept: Log-odds when $X = 0$; convert to probability with $\frac{e^{\beta_0}}{1+e^{\beta_0}}$
3. Slope: Change in log-odds per unit X; odds ratio $= e^{\beta_1}$
4. Decision boundary: Solve $\beta_0 + \beta_1 X = 0$ for X

Log-Log Regression

$$\log(Y) = \beta_0 + \beta_1 \log(X)$$

- Doubling X multiplies Y by 2^{β_1}
- Useful for multiplicative/financial relationships
- Often fixes heteroscedasticity

Hierarchical Models

- **Problem**: Grouped data (players, students, patients)
- **Solution**: Parameters vary by group, drawn from common distribution
- **Effect**: Shrinkage—extreme estimates pulled toward mean
- **Benefit**: Better estimates for groups with little data

Model: $y_{ij} \sim f(\alpha_j + \beta X_{ij})$, where $\alpha_j \sim N(\alpha_0, \sigma_\alpha^2)$

Bayesian Logistic Regression

- Put Normal priors on β coefficients (not Beta—wrong support!)
- Normal prior centered at 0 $\Leftrightarrow$ Ridge regularization
- No conjugacy: posterior doesn't have nice form
- Solution: MCMC simulation

Key Concepts

- **Conjugacy**: Prior + Likelihood = Same family posterior
- **Shrinkage**: Pulling estimates toward the mean
- **Hyperparameters**: Parameters of the prior distribution
- **Posterior predictive**: $\int f(\tilde{y}|\theta) f(\theta|\text{data}) d\theta$

Lecture 17: MCMC and Missing Data

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 17
- **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb
- **Topics:** MCMC (Markov Chain Monte Carlo), Metropolis-Hastings Algorithm, Missing Data, Imputation

Lecture Overview

This lecture covers two important but distinct topics in data science:

1. **MCMC (Markov Chain Monte Carlo):** When Bayesian posterior distributions don't have nice closed forms (like in logistic regression), we need numerical methods to sample from them. MCMC provides a powerful framework for this.
2. **Missing Data:** Real-world datasets often have missing values. How you handle missing data can dramatically affect your results. We'll learn principled approaches beyond simply dropping rows.

Key Takeaway: These methods help us deal with the messy reality of statistical modeling—complex posteriors that can't be solved analytically, and datasets that aren't complete.

Contents

1 Review: Why Do We Need MCMC?

1.1 The Problem with Non-Conjugate Posteriors

In previous lectures, we discussed Bayesian inference and the concept of **conjugacy**:

- **Normal-Normal:** Normal prior on mean μ with Normal likelihood gives Normal posterior
- **Gamma-Normal:** Gamma prior on precision $(1/\sigma^2)$ gives Gamma posterior
- **Beta-Binomial:** Beta prior on probability p with Binomial likelihood gives Beta posterior

Key Information

Why Conjugacy is Nice:
When the posterior has a known distribution family, we can:

- Write down the posterior analytically
- Easily sample from it using standard functions
- Compute means, variances, credible intervals directly

1.2 What Happens in Logistic Regression?

When we tried to apply Bayesian methods to logistic regression, things fell apart:

1. Our unknown parameters are α (intercept) and β (slope)
2. These are linked to probability through the logistic function:

$$p = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

3. We put Normal priors on α and β
4. The posterior is no longer a named distribution!

Warning

The Core Problem:
The logistic function is **nonlinear**, and this nonlinearity destroys the nice conjugate relationship. The Beta-Binomial-Beta pattern doesn't work when we transform through $\frac{e^x}{1+e^x}$.
The posterior is some complicated function that we can write down but can't identify as any standard distribution.

1.3 What Can We Still Do?

Even though we can't name the posterior distribution, we can:

- **Evaluate its height:** Given any (α, β), we can compute the posterior density (up to a normalizing constant)
- **Compare relative probabilities:** We can tell which (α, β) pairs are more likely

The question becomes: *How do we explore and summarize a distribution we can evaluate but can't*

sample from directly?

Key Summary

Key Insight:

The posterior is *proportional to* prior $\times$ likelihood:

$$\pi(\alpha, \beta|\text{data}) \propto \pi(\alpha) \cdot \pi(\beta) \cdot L(\text{data}|\alpha, \beta)$$

We know the right-hand side. We just don't know what distribution it forms.

2 Monte Carlo Methods: The Big Picture

2.1 What is Monte Carlo?

Definition:

Monte Carlo Methods **Monte Carlo methods** are computational algorithms that use random sampling to obtain numerical results. The idea is simple: if you want to understand a distribution, generate many random samples from it and use those samples to estimate properties like the mean, variance, or credible intervals.

Think of it like this: instead of solving a complex integral analytically, we:

1. Generate thousands of random samples from the distribution
2. Use the empirical distribution of samples to estimate whatever we need

Example:

Estimating π with Monte Carlo Imagine a square with a circle inscribed inside it:

- Square has side length 2 (area = 4)
- Circle has radius 1 (area = π)

To estimate π:

1. Randomly throw darts at the square (uniform random points)
2. Count how many land inside the circle
3. Ratio $\approx$ Circle area / Square area = $\pi/4$

If 78.5% of 10,000 darts land in the circle, we estimate $\pi \approx 4 \times 0.785 = 3.14$

2.2 Where Have We Seen This Before?

Monte Carlo principles appear throughout statistics:

- **Bootstrapping:** Resample data to estimate sampling distribution of statistics
- **Permutation testing:** Randomly permute labels to create null distribution
- **Cross-validation:** Randomly split data to estimate model performance

All of these use repeated random sampling to empirically estimate something we can't compute analytically.

3 Rejection Sampling

3.1 The Basic Idea

Rejection sampling is a technique for sampling from a **target distribution** when we can only easily sample from a different **proposal distribution**.

Definition:

Rejection Sampling **Goal:** Sample from target distribution $f(x)$ (e.g., our posterior)
Tool: We can sample from proposal distribution $g(x)$ (e.g., a Normal)
Requirement: $f(x) \leq M \cdot g(x)$ for all x and some constant M

The analogy: *We want to sample from the unit circle, but we can only sample from the unit square. So we sample from the square and reject points outside the circle.*

3.2 The Cookie Monster Example

Imagine Cookie Monster is analyzing cookies from a bakery with four types: chocolate chip, oatmeal raisin, peanut butter, and sugar. Each type has different size distributions:

- Chocolate chip: centered around 8 cm diameter
- Oatmeal raisin: centered around 10 cm
- Peanut butter: centered around 12 cm
- Sugar: centered around 14 cm

When we look at the **marginal distribution** of all cookies combined, we get a complex multimodal distribution (possibly 3-4 peaks visible).

How do we sample from this weird distribution?

3.3 The Algorithm

1. Choose a proposal distribution $g(x)$ (e.g., a single Normal) that we can sample from
2. Scale it up by constant M so that $M \cdot g(x) \geq f(x)$ everywhere
3. Repeat:
   - Draw x from $g(x)$
   - Draw u from Uniform(0, 1)
   - If $u < \frac{f(x)}{M \cdot g(x)}$, accept x; otherwise reject

Key Information

Why This Works:
- When $f(x)$ is close to $M \cdot g(x)$: We accept most of the time
- When $f(x)$ is much smaller than $M \cdot g(x)$: We reject most of the time

The acceptance probability is exactly $\frac{f(x)}{M \cdot g(x)}$, which ensures we end up with samples distributed

according to $f(x)$.

3.4 Practical Example

Suppose:

- Target (black curve): Complex multimodal cookie size distribution
- Proposal (red curve): Single Normal, scaled up by $M = 11$

At $x = 10$ (center):

- Black curve height: 0.05
- Red curve height: 0.55
- Acceptance probability: $0.05/0.55 \approx 9\%$

At $x = 14$ (tail where black is higher relative to red):

- Black curve height: 0.04
- Red curve height: 0.045
- Acceptance probability: $0.04/0.045 \approx 89\%$

Warning

The Inefficiency Problem:
If $M = 11$, on average only about 1 in 11 proposals are accepted!
To get 10,000 samples from the target, we need to generate about 110,000 samples from the proposal.
This gets even worse in high dimensions.

3.5 Using the Samples

Once we have enough accepted samples, we can:

- Plot the histogram $\rightarrow$ Visual representation of posterior
- Compute the mean $\rightarrow$ Posterior mean estimate
- Find the 2.5th and 97.5th percentiles $\rightarrow$ 95% credible interval

Just like in bootstrapping, we use the **empirical distribution** of samples to estimate properties of the **theoretical distribution**.

3.6 Limitations of Rejection Sampling

1. **Computational waste:** Many proposals are rejected
2. **Curse of dimensionality:** In high dimensions (e.g., logistic regression with 20 predictors = 21 parameters), finding a good proposal distribution is nearly impossible
3. **Scaling constant M:** Needs to be large enough to cover the target everywhere, leading to low acceptance rates

Key Summary

Rejection sampling is conceptually simple and works well in low dimensions, but it becomes impractical for complex, high-dimensional posteriors. We need a better approach: **MCMC**.

4 Markov Chain Monte Carlo (MCMC)

4.1 What is MCMC?

Definition:

MCMC **Markov Chain Monte Carlo (MCMC)** is a class of algorithms that sample from a probability distribution by constructing a Markov chain that has the desired distribution as its equilibrium distribution.

Let's break this down:

- **Monte Carlo:** Using random sampling
- **Markov Chain:** A sequence where each step depends only on the previous step

4.2 The Markovian Property

Definition:

Markov Property (Memorylessness) A process has the **Markov property** if the probability of being in a future state depends *only* on the current state, not on how we got there.

$$P(\theta_{t+1}|\theta_t, \theta_{t-1}, \ldots, \theta_0) = P(\theta_{t+1}|\theta_t)$$

Example:

Real-World Markov Chains **Weather:**

- Tomorrow's weather depends mainly on today's weather
- If it's raining today, there's a higher chance of rain tomorrow
- We don't need to know what happened last week

Stock Prices:

- Tomorrow's price depends on today's price
- The path that got us to today's price doesn't matter
- "Where you are" is what matters, not "how you got there"

4.3 The Key Insight of MCMC

Instead of sampling independently from the distribution (like rejection sampling), MCMC creates a **random walk** through the parameter space.

Imagine a robot walking around a landscape where height represents probability:

- Higher terrain = Higher posterior probability
- The robot tends to move uphill (toward higher probability)
- Sometimes it moves downhill (but less often)
- Over time, the robot spends more time in high-probability regions

Key Information

Why MCMC Works:

If we construct the random walk correctly, the **long-run frequency** of visits to any region will be proportional to the posterior probability of that region.

After enough steps, the sequence of positions forms a sample from the posterior distribution!

5 The Metropolis-Hastings Algorithm

5.1 Algorithm Overview

The **Metropolis-Hastings algorithm** is the most common MCMC method:

1. Start at some initial parameter value θ_0
2. At each step t:
   - Propose a new value θ^* based on current position θ_t
   - Decide whether to accept or reject the proposal
   - If accept: $\theta_{t+1} = \theta^*$
   - If reject: $\theta_{t+1} = \theta_t$ (stay in place)
3. Repeat many times

5.2 The Proposal Distribution

The **proposal distribution** $q(\theta^*|\theta_t)$ determines how we suggest new values:

Common choice: **Random walk proposal**

$$\theta^* = \theta_t + \epsilon, \quad \epsilon \sim \text{Normal}(0, \sigma^2)$$

This means: "Take a random step from current position"

- Small σ: Small steps, slow exploration
- Large σ: Big steps, might miss important regions

5.3 The Acceptance Probability

Here's the key formula—the probability of accepting a proposed move:

$$\alpha = \min\left(1, \frac{f(\theta^*)}{f(\theta_t)} \cdot \frac{q(\theta_t|\theta^*)}{q(\theta^*|\theta_t)}\right) \tag{1}$$

Where:

- $f(\theta)$ is the target distribution (posterior, up to normalizing constant)
- $q(\theta^*|\theta_t)$ is the proposal distribution

Key Information

Intuition Behind the Formula:
Ratio $\frac{f(\theta^*)}{f(\theta_t)}$:

- If θ^* has higher posterior probability → ratio > 1
- We accept with probability 1 (always move uphill!)
- If θ^* has lower probability → ratio < 1

- We accept with that probability (sometimes move downhill)

The proposal correction term $\frac{q(\theta_t|\theta^*)}{q(\theta^*|\theta_t)}$:

- For symmetric proposals (like random walk), this equals 1
- Needed for asymmetric proposals to ensure proper sampling

5.4 Step-by-Step Example

Suppose we're at θ_t where $f(\theta_t) = 0.5$

We propose θ^* where $f(\theta^*) = 0.3$ (downhill move)

Acceptance probability (with symmetric proposal):

$$\alpha = \min\left(1, \frac{0.3}{0.5}\right) = 0.6$$

We generate $u \sim \text{Uniform}(0, 1)$:

- If $u < 0.6$: Accept, move to θ^*
- If $u \geq 0.6$: Reject, stay at θ_t

Key Summary

Key Properties of Metropolis-Hastings:

1. **Uphill moves:** Always accepted
2. **Downhill moves:** Sometimes accepted (proportional to ratio)
3. **Result:** Spends more time in high-probability regions
4. **Doesn't require normalizing constant:** Only needs ratios!

6 Practical MCMC: The Skittles Example

6.1 The Problem Setup

A candy company wants to introduce a new Skittles flavor: Mango. They need to determine the optimal amount of "mango essence" ingredient.

Data collection:

- Different dosages (amounts of mango essence) are tested
- Multiple taste testers try each dosage
- Each tester says "Yes, I love it" or "No"

This is a classic **dose-response** problem!

6.2 The Model

- **Predictor x:** Amount of mango essence (mg)
- **Response y:** Number of people who love it out of n testers
- **Distribution:** $Y \sim \text{Binomial}(n, p(x))$

We link p to x through logistic regression:

$$\log\left(\frac{p}{1-p}\right) = \alpha + \beta x$$

Or equivalently:

$$p = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

6.3 Bayesian Setup

Unknown parameters: α (intercept) and β (slope)

Priors:

$$\alpha \sim \text{Normal}(0, 100^2)$$
$$\beta \sim \text{Normal}(0, 100^2)$$

Why Normal(0, 100)?

- Centered at 0: No prior assumption about direction of effect
- Large variance (100): Very non-informative, letting data speak
- On log-odds scale, α and β can be any real number

Posterior:

$$\pi(\alpha, \beta|\text{data}) \propto \pi(\alpha) \cdot \pi(\beta) \cdot \prod_{i=1}^{n} \binom{n_i}{y_i} p_i^{y_i}(1-p_i)^{n_i-y_i}$$

This is a complicated function—not a named distribution!

6.4 Using PyMC for MCMC

```python
import pymc as pm
import numpy as np

# Data
flavoring = np.array([1.88, 1.86, 1.84, 1.82, ...])  # Dosages
y = np.array([60, 61, 58, 56, ...])  # Number who loved it
n = np.array([60, 62, 60, 58, ...])  # Total testers

with pm.Model() as skittles_model:
    # Priors
    alpha = pm.Normal('alpha', mu=0, sigma=100)
    beta = pm.Normal('beta', mu=0, sigma=100)

    # Logistic transformation
    p = pm.math.invlogit(alpha + beta * flavoring)

    # Likelihood
    y_obs = pm.Binomial('y_obs', n=n, p=p, observed=y)

    # MCMC Sampling
    trace = pm.sample(2000, tune=2000, return_inferencedata=True)
```

Key parameters:

- `tune=2000`: Burn-in period (samples to discard)
- `2000`: Number of samples to keep after burn-in

6.5 Interpreting the Results

After running MCMC, we get a **trace**—a sequence of sampled (α, β) pairs.

Trace plots show the sampled values over time:

- Should look like "fuzzy caterpillars" or "hairy worms"
- No trends or patterns
- Bouncing randomly around a stable mean

Warning

Signs of Problems in Trace Plots:

- Gradual drift upward or downward
- Getting stuck in one place for many iterations
- Different chains showing different patterns

These indicate the chain hasn't converged to the posterior!

Posterior histograms show the distribution of samples:

- For β: Values range from about 25 to 42

- All positive! This tells us more mango essence → more people love it
- Posterior mean: around 33-34
- 95% credible interval: approximately [28, 40]

7 MCMC Diagnostics

7.1 The Burn-in Period

Definition:

Burn-in (Warm-up) The **burn-in** period is the initial portion of the MCMC chain that is discarded before collecting samples. During burn-in, the chain is "warming up" and moving from its arbitrary starting point toward the high-probability region.

Why is burn-in necessary?

- We start the chain at some arbitrary initial value
- If we start in a low-probability region, early samples don't represent the posterior
- We wait until the chain "finds" the high-probability region

7.2 The R-hat Statistic ($\hat{R}$)

Definition:

R-hat ($\hat{R}$) **R-hat** (also called the potential scale reduction factor) measures whether multiple MCMC chains have converged to the same distribution.

$$\hat{R} \approx 1.0 \quad \text{indicates convergence}$$

How it works:

1. Run multiple independent chains (typically 4)
2. Compare the variance *within* each chain to variance *between* chains
3. If chains have converged, these should be similar ($\hat{R} \approx 1$)
4. If chains are exploring different regions, between-chain variance is larger ($\hat{R} > 1$)

Important:

Convergence Rules

- $\hat{R} < 1.01$: Excellent convergence
- $\hat{R} < 1.05$: Generally acceptable
- $\hat{R} > 1.1$: **Serious problem!** Do not trust results!

If $\hat{R} > 1.1$, try:

1. Increase burn-in period
2. Run chains longer
3. Try different initial values
4. Reparameterize the model

7.3 Effective Sample Size (ESS)

Because MCMC samples are **correlated** (each depends on the previous), they don't provide as much information as independent samples.

Definition:

Effective Sample Size The **effective sample size (ESS)** is the equivalent number of independent samples that would provide the same information as the correlated MCMC samples.
If you have 2000 MCMC samples but ESS = 500, your samples are as informative as 500 independent samples.

The MCMC standard error for the mean is:

$$\text{MCSE} = \frac{\text{Posterior SD}}{\sqrt{\text{ESS}}}$$

Not $\sqrt{n}$ where n is the number of MCMC samples!

7.4 Summary Statistics from PyMC

Typical output includes:

- **mean**: Posterior mean estimate
- **sd**: Posterior standard deviation
- **hdi_3%**, **hdi_97%**: 94% credible interval bounds
- **mcse_mean**: Monte Carlo standard error for the mean
- **ess_bulk**: Effective sample size
- **r_hat**: Convergence diagnostic

Key Summary

MCMC Summary:

1. MCMC constructs a random walk through parameter space
2. After burn-in, samples represent the posterior distribution
3. Always check diagnostics: trace plots, $\hat{R}$, ESS
4. Use posterior samples to compute means, credible intervals, predictions

8 Missing Data: Introduction

8.1 What is Missing Data?

In real-world datasets, values are often missing:

- Survey respondents skip questions
- Sensors fail to record measurements
- Data entry errors leave blanks
- Information is not collected for some subjects

How does Python handle missing values?

- Pandas represents missing as `NaN` (Not a Number) or `NA`
- Scikit-learn **throws errors** if given data with `NaN`
- You must handle missing data before modeling!

8.2 Common (But Problematic) Approaches

Option 1: Drop rows with missing values

```
df_clean = df.dropna()  # Remove rows with any NaN
```

Option 2: Drop columns with missing values

```
df_clean = df.dropna(axis=1)  # Remove columns with any NaN
```

Option 3: Fill with mean/median

```
df['column'] = df['column'].fillna(df['column'].mean())
```

Warning

Problems with Simple Approaches:

Dropping rows:

- Loses valuable data
- Can introduce **bias** if missingness is related to the outcome
- Reduces sample size and statistical power

Mean imputation:

- **Underestimates variance** (makes distribution narrower)
- Distorts relationships between variables
- Doesn't account for uncertainty in imputed values

9 Types of Missing Data

Understanding *why* data is missing is crucial for choosing the right approach.

9.1 Missing Completely at Random (MCAR)

Definition:

MCAR Data is **Missing Completely at Random** if the probability of being missing is unrelated to:

- The missing value itself
- Any other observed variables

Missingness is purely random, like someone randomly punching holes in your spreadsheet.

Example: A survey is accidentally not sent to 5% of randomly selected participants.

Implication: Complete case analysis (dropping rows) gives unbiased estimates, just with reduced sample size.

9.2 Missing at Random (MAR)

Definition:

MAR Data is **Missing at Random** if the probability of being missing can be fully explained by other *observed* variables (but not by the missing value itself).

Example: In a depression study, younger people are less likely to report their income. The missingness is related to age (which we observe), not to income itself.

Implication: We can use observed variables to model the missingness and correct for bias.

9.3 Missing Not at Random (MNAR)

Definition:

MNAR Data is **Missing Not at Random** if the probability of being missing depends on the unobserved value itself or on unobserved variables.

Example: People with very high incomes are less likely to report their income *because* it's high.

Implication: This is the hardest case. No statistical method can fully correct for the bias without strong assumptions.

Important:

The Fundamental Problem We can never know for certain which type of missingness we have. The classification depends on information we don't observe. This is why missing data handling requires careful thought and often sensitivity analysis.

10 Better Approaches to Missing Data

10.1 The Indicator Variable Method

A simple but often effective approach:

1. Create a new variable indicating whether the original value was missing
2. Impute a constant (like 0) for missing values
3. Include **both** the imputed variable and the indicator in your model

Example:

Indicator Method Original data:

ID	Income	Outcome
1	50000	1
2	NaN	0
3	75000	1
4	NaN	1

After transformation:

ID	Income_imputed	Income_missing	Outcome
1	50000	0	1
2	0	1	0
3	75000	0	1
4	0	1	1

Why this works:

- The imputed zeros aren't treated as real zeros
- The indicator variable captures the "missingness effect"
- The model can learn different relationships for missing vs. non-missing cases

```python
import pandas as pd
import numpy as np

def add_missing_indicator(df, column):
    """Add missing indicator and impute zeros"""
    # Create indicator
    df[f'{column}_missing'] = df[column].isna().astype(int)
    # Impute with 0
    df[f'{column}_imputed'] = df[column].fillna(0)
    return df

# Usage
df = add_missing_indicator(df, 'income')
# Now use both 'income_imputed' and 'income_missing' in model
```

10.2 Categorical Variables

For categorical variables, treat "missing" as its own category:

Example:

Missing as a Category Original variable `color` with values: Red, Blue, Green, NaN
After transformation: Red, Blue, Green, Missing
This is equivalent to one-hot encoding with three original categories plus one missing category.

10.3 Multiple Imputation (Advanced)

The gold standard for handling missing data:

1. Create multiple (5-20) imputed datasets, each with slightly different imputed values
2. Analyze each dataset separately
3. Combine results using special rules that account for imputation uncertainty

This is more complex but properly accounts for uncertainty about the missing values.

11 Key Takeaways

Key Summary

MCMC and Missing Data: Summary

MCMC:

- When posteriors don't have closed forms, MCMC lets us sample from them
- Metropolis-Hastings: Propose → Accept/Reject → Repeat
- Always check convergence: trace plots, $\hat{R} \approx 1.0$, adequate ESS
- Use samples to estimate means, credible intervals, predictions

Missing Data:

- Three types: MCAR, MAR, MNAR (we usually don't know which)
- Simple dropping or mean imputation can bias results
- Better: Indicator variable method (impute + flag)
- Best: Multiple imputation (accounts for uncertainty)

The Common Theme: Both topics deal with **uncertainty**—uncertainty in posterior distributions (MCMC) and uncertainty about missing values (imputation). Good data science acknowledges and properly accounts for uncertainty.

Table 1: *MCMC vs. Rejection Sampling Comparison*

Aspect	Rejection Sampling	MCMC
Sample independence	Independent samples	Correlated samples
Efficiency in high-D	Very poor	Much better
Requires	Envelope distribution $M \cdot g(x)$	Proposal distribution q
Acceptance rate	Can be very low	Tunable (aim for 20-50%)
Burn-in needed	No	Yes
Convergence diagnostics	Not applicable	Essential ($\hat{R}$, ESS)

Table 2: *Missing Data Handling Methods*

Method	Pros	Cons
Drop rows (listwise deletion)	Simple, unbiased if MCAR	Loses data, biased if not MCAR
Mean imputation	Simple, preserves sample size	Underestimates variance, biased
Indicator variable method	Captures missingness pattern	Assumes missingness is informative
Multiple imputation	Proper uncertainty quantification	Complex, computationally intensive

12 Python Code Reference

12.1 MCMC with PyMC

```
import pymc as pm
import arviz as az  # For diagnostics
```

```
3
4  # Define model
5  with pm.Model() as model:
6      # Priors
7      alpha = pm.Normal('alpha', mu=0, sigma=10)
8      beta = pm.Normal('beta', mu=0, sigma=10)
9
10     # Likelihood (logistic regression example)
11     p = pm.math.invlogit(alpha + beta * X)
12     y_obs = pm.Binomial('y_obs', n=n, p=p, observed=y)
13
14     # Sample
15     trace = pm.sample(2000, tune=2000, cores=4,
16                       return_inferencedata=True)
17
18 # Diagnostics
19 az.plot_trace(trace)  # Trace plots
20 az.summary(trace)     # Summary statistics including r_hat
```

12.2 Missing Data Handling

```
1  import pandas as pd
2  import numpy as np
3  from sklearn.impute import SimpleImputer
4
5  # Method 1: Indicator variable approach
6  def handle_missing_with_indicator(df, columns):
7      for col in columns:
8          df[f'{col}_missing'] = df[col].isna().astype(int)
9          df[col] = df[col].fillna(0)  # or fillna(df[col].mean())
10     return df
11
12 # Method 2: Using sklearn
13 imputer = SimpleImputer(strategy='mean')  # or 'median', 'most_frequent'
14 X_imputed = imputer.fit_transform(X)
15
16 # Check missingness pattern
17 print(df.isna().sum())  # Count NaN per column
18 print(df.isna().mean())  # Proportion NaN per column
```

13 Checklist for Understanding

☐ Can you explain why we need MCMC when the posterior isn't a named distribution?

☐ Can you describe the basic idea of Metropolis-Hastings (propose, accept/reject)?

☐ Do you understand what trace plots should look like for a converged chain?

☐ Can you interpret $\hat{R}$ and know when it indicates problems?

☐ Do you know the three types of missing data (MCAR, MAR, MNAR)?

☐ Can you implement the indicator variable method for handling missing data?

☐ Do you understand why simple mean imputation can be problematic?

Lecture 18: Decision Trees for Classification

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 18
- **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb
- **Topics:** Decision Trees, Splitting Criteria, Gini Index, Entropy, Stopping Conditions, Overfitting

Lecture Overview

This lecture introduces **Decision Trees**, a fundamentally different approach to classification compared to logistic regression. Decision trees are intuitive, interpretable, and can model complex decision boundaries.

Key Topics:

1. Why we need decision trees (limitations of logistic regression)
2. How decision trees make predictions (tree traversal)
3. How decision trees learn (splitting criteria)
4. How to prevent overfitting (stopping conditions)

Key Insight: Decision trees work like a game of "20 Questions"—a series of yes/no questions leads to a final classification. This simple idea creates surprisingly powerful models that can handle non-linear decision boundaries.

Contents

1 Motivation: Why Not Just Use Logistic Regression?

1.1 What Logistic Regression Does Well

Logistic regression works beautifully when:

- Classes are **linearly separable** (or close to it)
- The **decision boundary** can be expressed as a straight line (or hyperplane)

Example:

Logistic Regression Success Consider classifying land as "agricultural" (green) vs "dry" (white) based on longitude and latitude.
If the data looks like this:

- Green points clustered on the right
- White points clustered on the left

A simple linear decision boundary works perfectly: $\text{latitude} = -0.8 \times \text{longitude} + 0.1$

1.2 Where Logistic Regression Fails

The problem arises when the decision boundary is **not linear**:

Example:

Logistic Regression Failure **Case 1: Circular boundary**

- Green points (agricultural) in the center
- White points (dry land) surrounding them in a ring

No straight line can separate these! You'd need a circle: $x_1^2 + x_2^2 < r^2$
Case 2: Scattered regions

- Green points in the top-left and bottom-right corners
- White points in the top-right and bottom-left corners

Now you'd need diagonal lines or complex polynomial boundaries!

Warning

The Polynomial Solution Has Limits
Yes, you could use polynomial logistic regression (adding x^2, $x_1 x_2$ terms) to create curved boundaries. But:

- It requires manually engineering features
- Complex boundaries need very high-degree polynomials
- You don't know what shape the boundary should be beforehand

1.3 Our Wish List for a New Model

We want a model that can:

1. **Create complex decision boundaries** without manually engineering features

2. Be **easy to interpret**—we can explain why the model made a decision
3. Be **computationally efficient** to train and predict

Key Summary

Enter Decision Trees!

Decision trees satisfy all three criteria:

- Complex boundaries through recursive splitting
- Interpretable through a flowchart-like structure
- Efficient through simple comparison operations

2 The Intuition: Flowcharts and 20 Questions

2.1 Everyday Decision Making

We make tree-like decisions every day:

- "Is it raining? If yes, take umbrella. If no, check temperature..."
- "Did I finish my homework? If yes, go to party. If no, how much time left?..."

Example:

The Engineering Flowchart Classic problem-solving flowchart:

Question 1: Does it move?

- **Yes** → Question 2: Should it move?
  - Yes → *No problem!*
  - No → *Use duct tape*
- **No** → Question 2: Should it move?
  - Yes → *Use WD-40*
  - No → *No problem!*

This is exactly how a decision tree works! Binary questions leading to final decisions.

2.2 Key Properties of This Approach

1. **Binary decisions:** Each question has only two answers (yes/no)
2. **Interpretable:** You can explain any decision by tracing the path
3. **Simple:** No distributions, no gradients—just comparisons
4. **Flexible:** The shape of the final regions can be arbitrarily complex

3 Decision Tree Terminology

Before diving deeper, let's establish the vocabulary:

Definition:

Tree Components

- **Root Node:** The topmost node where the tree starts (first question)
- **Internal Nodes:** Nodes that ask questions and split into children
- **Leaf Nodes (Terminal Nodes):** Final nodes that make predictions (no children)
- **Split:** The act of dividing a node into child nodes based on a question
- **Depth:** The number of splits from root to a node
- **Branch:** The path from root to any node

Key Information

Visualizing the Tree:

Decision trees are typically drawn "upside down":

- Root at the top (like a family tree)
- Leaves at the bottom
- Data flows downward through questions

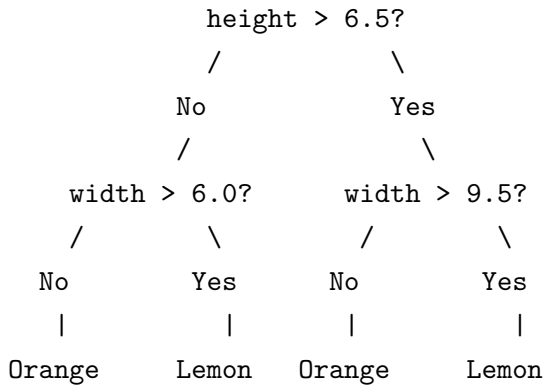
4 How Decision Trees Make Predictions

4.1 The Lemon vs Orange Example

Suppose we have a trained decision tree that classifies fruits as "Lemon" or "Orange" based on:

- `height`: Height of the fruit
- `width`: Width of the fruit

The tree structure:



4.2 Prediction: Tree Traversal

Definition:

Tree Traversal **Tree traversal** is the process of starting at the root node and following the appropriate branches based on feature values until reaching a leaf node, which provides the prediction.

Example:

Predicting a New Fruit A new fruit arrives with measurements: `height = 5.9`, `width = 5.8`

Step 1: Root node asks: "Is height > 6.5?"

- 5.9 is NOT greater than 6.5 $\rightarrow$ Go **left** (No branch)

Step 2: Next node asks: "Is width > 6.0?"

- 5.8 is NOT greater than 6.0 $\rightarrow$ Go **left** (No branch)

Step 3: We've reached a leaf node labeled "Orange"

Prediction: This fruit is an **Orange**

4.3 Decision Boundaries in Feature Space

Each split in the tree creates a boundary that is **parallel to a feature axis**:

- Split on `height > 6.5` creates a **horizontal line** at height = 6.5
- Split on `width > 6.0` creates a **vertical line** at width = 6.0

The final decision regions are **axis-aligned rectangles**:

- Region 1: height ≤ 6.5 AND width $\leq 6.0 \rightarrow$ Orange
- Region 2: height ≤ 6.5 AND width $> 6.0 \rightarrow$ Lemon

- Region 3: height > 6.5 AND width $\leq 9.5 \rightarrow$ Orange
- Region 4: height > 6.5 AND width $> 9.5 \rightarrow$ Lemon

Key Information

Why Rectangles?

Decision trees always create **axis-aligned rectangular regions** because:

1. Each split is on a single feature
2. Splits create boundaries perpendicular to that feature's axis
3. Multiple splits partition space into nested rectangles

This is different from logistic regression, which creates tilted linear boundaries.

5 How Decision Trees Learn: Splitting Criteria

5.1 The Learning Problem

We've seen how to *use* a decision tree. But how do we *build* one?

The key questions:

1. Which feature should we split on?
2. What threshold value should we use?
3. When should we stop splitting?

5.2 The Goal: Maximize Purity

Definition:

Purity and Impurity

- A node is **pure** if all its data points belong to the same class
- A node is **impure** if it contains a mixture of classes
- **Impurity** measures how "mixed" a node is

The goal of each split is to create child nodes that are **more pure** than the parent.

Example:

Good vs Bad Splits **Parent node:** 6 blue circles, 8 orange triangles (14 total)

Split A:

- Left child: 6 blue, 0 orange (100% pure!)
- Right child: 0 blue, 8 orange (100% pure!)

→ **Excellent split!** Both children are perfectly pure.

Split B:

- Left child: 3 blue, 4 orange (mixed)
- Right child: 3 blue, 4 orange (mixed)

→ **Terrible split!** Children are just as impure as parent.

5.3 Measuring Impurity: Three Approaches

5.3.1 Method 1: Classification Error

The most intuitive measure: what fraction would we get wrong if we predicted the majority class?

$$\text{Classification Error} = 1 - \max_k \hat{p}_k \tag{1}$$

Where $\hat{p}_k$ is the proportion of class k in the node.

Example:

Classification Error Calculation Node with 5 blue circles and 8 orange triangles (13 total):

- $\hat{p}_{\text{blue}} = 5/13 = 0.385$
- $\hat{p}_{\text{orange}} = 8/13 = 0.615$

If we predict "orange" (majority), we get 5 wrong.
Classification Error $= 1 - \max(0.385, 0.615) = 1 - 0.615 = 0.385$

5.3.2 Method 2: Gini Impurity (Index)

The most commonly used measure in practice:

$$\text{Gini} = 1 - \sum_{k=1}^{K} \hat{p}_k^2 \tag{2}$$

Definition:

Interpreting Gini Gini impurity represents the probability that two randomly chosen samples from the node would have **different** class labels.

- **Gini = 0:** Node is perfectly pure (all same class)
- **Gini = 0.5:** Maximum impurity for 2 classes (50-50 split)
- For K classes: Maximum Gini $= 1 - 1/K$

Example:

Gini Calculation Same node: 5 blue, 8 orange (13 total)
$\text{Gini} = 1 - \left[\left(\frac{5}{13}\right)^2 + \left(\frac{8}{13}\right)^2\right]$
$= 1 - \left[\frac{25}{169} + \frac{64}{169}\right] = 1 - \frac{89}{169} = \frac{80}{169} \approx 0.47$
Compare to a pure node (all orange): $\text{Gini} = 1 - [0^2 + 1^2] = 0$

5.3.3 Method 3: Entropy

From information theory, entropy measures the "uncertainty" or "disorder":

$$\text{Entropy} = -\sum_{k=1}^{K} \hat{p}_k \log_2(\hat{p}_k) \tag{3}$$

Definition:

Interpreting Entropy Entropy measures the number of bits needed to encode the class of a random sample.

- **Entropy = 0:** Node is perfectly pure (no uncertainty)
- **Entropy = 1:** Maximum impurity for 2 classes (1 bit needed)
- For K classes: Maximum Entropy $= \log_2(K)$

Example:

Entropy Calculation Same node: 5 blue, 8 orange

Entropy $= -\left[\frac{5}{13}\log_2\left(\frac{5}{13}\right) + \frac{8}{13}\log_2\left(\frac{8}{13}\right)\right]$

$= -[0.385 \times (-1.38) + 0.615 \times (-0.70)]$

$= -[-0.53 - 0.43] = 0.96$

Compare to a pure node: Entropy $= -[1 \times \log_2(1)] = -[1 \times 0] = 0$

5.4 Why Squaring in Gini?

The squaring operation in Gini impurity has an important effect:

Key Information

Effect of Squaring:

Squaring the proportions **accentuates the difference** between pure and impure regions:

- Large proportions (majority class) contribute more when squared
- Small proportions (minority classes) are diminished when squared
- This makes Gini more sensitive to "almost pure" vs "quite mixed"

Think of it like **softmax**—it emphasizes the maximum without explicitly computing max!

5.5 Comparing the Three Measures

All three measures:

- Equal 0 when the node is pure
- Reach maximum when classes are evenly split
- Are monotonically related to purity

Key difference: Sensitivity to impurity changes

Table 1: *Impurity Measures Comparison*

Property	Classification Error	Gini Impurity	Entropy
Value at pure node	0	0	0
Max value (2 classes)	0.5	0.5	1.0
Sensitivity to changes	Least sensitive	Moderately sensitive	Most sensitive
Computational cost	Lowest	Low	Slightly higher
Default in sklearn	No	**Yes**	No

Warning

Why Not Use Classification Error?

Classification error is the most intuitive but the **least sensitive** to small improvements in purity.

Consider: A node goes from (50%, 50%) to (60%, 40%)

- Classification error: $0.5 \rightarrow 0.4$ (decrease of 0.1)

- Gini: $0.5 \rightarrow 0.48$ (slight decrease)

Gini and Entropy have **curved shapes** that penalize impurity more heavily near 50-50, making them better for finding good splits.

5.6 Evaluating a Split: Weighted Average

When evaluating a split, we must consider the **sizes** of the resulting child nodes.

Definition:

Weighted Impurity For a split creating regions R_1 (with N_1 samples) and R_2 (with N_2 samples):

$$\text{Weighted Impurity} = \frac{N_1}{N} \times \text{Impurity}(R_1) + \frac{N_2}{N} \times \text{Impurity}(R_2) \tag{4}$$

where $N = N_1 + N_2$ is the total samples.

Example:

Why Weighted Average Matters Consider two splits of a node with 18 samples:

Split A:

- R_1: 1 sample, Gini = 0 (pure)
- R_2: 17 samples, Gini = 0.45 (impure)

Weighted: $\frac{1}{18}(0) + \frac{17}{18}(0.45) = 0.425$

Split B:

- R_1: 9 samples, Gini = 0.2
- R_2: 9 samples, Gini = 0.2

Weighted: $\frac{9}{18}(0.2) + \frac{9}{18}(0.2) = 0.2$

Split B is better! Even though Split A has a "pure" child, it only contains 1 sample.

6 The Learning Algorithm: Greedy Approach

6.1 Why Can't We Find the "Best" Tree?

Finding the globally optimal decision tree is **NP-complete**—computationally infeasible for any reasonable dataset.

Why? Consider:

- p features, each with many possible split points
- At each node, we choose one split
- The number of possible trees grows **exponentially**

6.2 The Greedy Solution

Definition:

Greedy Algorithm A **greedy algorithm** makes the locally optimal choice at each step, without considering future consequences. It doesn't guarantee a globally optimal solution but is computationally tractable.

Decision Tree Learning Algorithm:

1. **Start:** All training data in root node
2. **For each node:**
   - For each feature p
   - For each possible threshold t
   - Calculate weighted impurity of split (p, t)
3. **Choose:** The (p^*, t^*) that **minimizes weighted impurity**
4. **Split:** Create two child nodes based on (p^*, t^*)
5. **Recurse:** Repeat steps 2-4 for each child
6. **Stop:** When stopping criterion is met

Key Information

Key Insight:
Unlike linear/logistic regression where we minimize a single loss function over all data, decision trees:

- Don't have a global loss function
- Make locally optimal decisions at each split
- This greedy approach is fast but may miss globally better trees

7 Preventing Overfitting: Stopping Conditions

7.1 The Overfitting Problem

If we let the tree grow without limits, it will keep splitting until:

- Every leaf node contains exactly **one** training sample
- Training accuracy is 100%

This is **severe overfitting**:

- The tree memorizes every training point, including noise
- It creates tiny rectangular regions around individual points
- Test accuracy will be poor

Warning

Visualizing Overfitting:
Imagine the true pattern is a circle of green points surrounded by white points.
Shallow tree (max_depth=4):

- Creates a rough square approximating the circle
- High bias, low variance
- Underfitting

Deep tree (max_depth=100):

- Creates tiny squares around every green point
- Even captures isolated green points in "white" regions (noise!)
- Low bias, high variance
- Overfitting

7.2 Stopping Conditions (Hyperparameters)

To prevent overfitting, we **intentionally limit** tree growth:

Definition:

Common Stopping Conditions

1. `max_depth`: Maximum number of splits from root to any leaf
   - `max_depth=3` means at most 3 questions to reach a prediction
   - Most commonly used regularization
2. `min_samples_leaf`: Minimum samples required in a leaf node
   - `min_samples_leaf=5` means stop if split would create a leaf with < 5 samples
   - Prevents very specific rules based on few points
3. `max_leaf_nodes`: Maximum total number of leaf nodes
   - `max_leaf_nodes=8` limits the tree to 8 final regions
   - Controls overall model complexity

4. `min_impurity_decrease`: Minimum impurity improvement required
   - Only split if it reduces impurity by at least this amount
   - Prevents splits that barely improve purity
5. **Pure nodes**: Automatically stop if a node is already 100% pure

7.3 Tree Growth Strategies

7.3.1 Level-Order Growth (Default)

The standard approach:

- Split all nodes at depth 1
- Then split all nodes at depth 2
- Continue level by level

Used with `max_depth` and most stopping conditions.

7.3.2 Best-First Growth

Alternative approach (used when `max_leaf_nodes` is set):

- At each step, consider ALL current leaf nodes
- Split the one that gives the **greatest impurity reduction**
- Regardless of its depth

Warning

Computational Cost of Best-First:
Best-first growth is more computationally expensive because:

- Must evaluate all possible splits at all leaf nodes
- Compare across different depths
- Number of comparisons grows with tree size

Use level-order when possible for efficiency.

7.4 Bias-Variance Tradeoff

Tree depth directly affects the bias-variance tradeoff:

7.5 Finding Optimal Hyperparameters

How do we choose the right `max_depth` or `min_samples_leaf`?

Answer: Cross-Validation!

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

```

Table 2: *Tree Depth and Bias-Variance*

	Shallow Tree	**Deep Tree**
Complexity	Low (simple model)	High (complex model)
Bias	High (underfits)	Low (fits training data well)
Variance	Low (stable across datasets)	High (changes with different data)
Training Error	Higher	Lower (approaching 0)
Test Error	May be high (underfitting)	May be high (overfitting)
Interpretability	High (few rules)	Low (many rules)

```
# Try different max_depth values
for depth in [2, 3, 4, 5, 6, 8, 10, None]:
    tree = DecisionTreeClassifier(max_depth=depth, random_state=42)
    scores = cross_val_score(tree, X, y, cv=5)
    print(f"max_depth={depth}: CV accuracy = {scores.mean():.3f}")

# Choose the depth with highest CV accuracy
```

8 Decision Trees in Python

8.1 Basic Usage

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create and train model
tree = DecisionTreeClassifier(
    criterion='gini',       # 'gini' or 'entropy'
    max_depth=5,            # Maximum depth
    min_samples_leaf=5,     # Minimum samples in leaf
    random_state=42
)
tree.fit(X_train, y_train)

# Predict
y_pred = tree.predict(X_test)
y_prob = tree.predict_proba(X_test)  # Probability estimates

# Evaluate
from sklearn.metrics import accuracy_score
print(f"Accuracy: {accuracy_score(y_test, y_pred):.3f}")
```

8.2 Visualizing the Tree

```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Plot the tree
plt.figure(figsize=(20, 10))
plot_tree(tree,
          feature_names=X.columns,
          class_names=['Orange', 'Lemon'],
          filled=True,
          rounded=True)
plt.title("Decision Tree Visualization")
plt.show()
```

8.3 Feature Importance

```python
# Get feature importances
importances = tree.feature_importances_
```

```

# Display
for name, importance in zip(X.columns, importances):
    print(f"{name}: {importance:.3f}")

# Most important feature = used in highest splits or most impurity reduction
```

9 Key Takeaways

Key Summary

Decision Trees Summary

What they are:

- Flowchart-like models that make binary decisions at each node
- Predictions by traversing from root to leaf
- Decision boundaries are axis-aligned rectangles

How they learn:

- Greedy algorithm: find best split at each step
- Goal: maximize purity (minimize impurity)
- Criteria: Gini impurity (default), Entropy, or Classification Error
- Evaluate splits using weighted average impurity

How to prevent overfitting:

- Stopping conditions: `max_depth`, `min_samples_leaf`, etc.
- Find optimal values via cross-validation

Advantages:

- Highly interpretable ("white box" model)
- No feature scaling needed
- Handle nonlinear boundaries
- Fast training and prediction

Limitations:

- Can only create axis-aligned boundaries
- Prone to overfitting without proper constraints
- Greedy algorithm may miss global optimum
- High variance (sensitive to training data)

10 Learning Checklist

☐ Can you explain why logistic regression fails for certain decision boundaries?

☐ Can you define: root node, internal node, leaf node, split, depth?

☐ Can you walk through tree traversal to make a prediction?

☐ Do you understand why decision boundaries are rectangular?

☐ Can you calculate Gini impurity for a given node?

☐ Can you calculate Entropy for a given node?

☐ Do you know why we use weighted average when evaluating splits?

☐ Can you explain why Gini/Entropy are preferred over classification error?

☐ Do you understand what "greedy" means in this context?

☐ Can you list 3+ stopping conditions and explain their purpose?

☐ Do you understand the bias-variance tradeoff as tree depth changes?

☐ Do you know how to find optimal hyperparameters using cross-validation?

11 Looking Ahead

In the next lectures, we'll extend decision trees to:

1. **Regression Trees**: Predicting continuous values instead of classes
2. **Random Forests**: Combining many trees to reduce variance
3. **Bagging and Boosting**: Ensemble methods for better performance
4. **Gradient Boosting**: XGBoost, LightGBM—state-of-the-art for tabular data

Lecture 19: Regression Trees and Pruning

CS109A: Introduction to Data Science

Harvard University

■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 19
■ **Instructors:**	Pavlos Protopapas, Kevin Rader, Chris Gumb
■ **Topics:**	Regression Trees, MSE as Splitting Criterion, Categorical Variables, Pruning, Cost Complexity

Lecture Overview

This lecture extends decision trees from classification to **regression** and introduces **pruning** as a powerful technique for controlling overfitting.

Key Topics:

1. **Regression Trees:** Using decision trees to predict continuous outcomes (instead of class labels)
2. **Categorical Variables:** Handling non-numeric features in decision trees
3. **Pruning:** A post-hoc approach to reducing tree complexity and preventing overfitting

Key Insight: The core ideas from classification trees carry over directly—we just swap impurity measures (Gini, Entropy) for **MSE**, and swap majority voting for **averaging**.

Contents

1 Regression Trees: From Classification to Prediction

1.1 Recap: Classification Trees

In classification trees, we learned:

- **Goal:** Predict a categorical outcome (e.g., "lemon" or "orange")
- **Splitting criterion:** Minimize impurity (Gini index or Entropy)
- **Prediction:** Majority vote—predict the most common class in the leaf node

1.2 What Changes for Regression?

For **regression trees**, we want to predict a **continuous** outcome (e.g., house price, temperature, stock return).

Definition:

Regression Tree A **regression tree** is a decision tree where:

- The target variable y is **continuous** (not categorical)
- Each leaf node predicts the **mean** of the training samples in that region
- Splits are chosen to minimize **MSE** (Mean Squared Error) instead of impurity

Table 1: *Classification vs Regression Trees*

Aspect	Classification Tree	Regression Tree
Target variable	Categorical (classes)	Continuous (numbers)
Splitting criterion	Gini impurity or Entropy	MSE (Mean Squared Error)
Prediction at leaf	Majority class	Mean of samples
Example prediction	"Survived" or "Did not survive"	"$450,000" or "25.3 degrees"

1.3 The Splitting Criterion: Minimizing MSE

In classification, we wanted each region to be "pure"—containing mostly one class.

In regression, we want each region to have **low variance**—containing samples with similar y values.

Definition:

MSE as Splitting Criterion For a region R with n samples, the MSE is:

$$\text{MSE}(R) = \frac{1}{n}\sum_{i \in R}(y_i - \bar{y}_R)^2 \tag{1}$$

Where $\bar{y}_R$ is the mean of all y values in region R.
This is simply the **variance** of y in that region!

Key Information

Why MSE Makes Sense:

If all samples in a region have similar y values:

- The mean $\bar{y}_R$ represents them well
- MSE is low (samples are close to the mean)
- Predicting $\bar{y}_R$ for new samples will be accurate

If samples have very different y values:

- The mean doesn't represent any sample well
- MSE is high (samples are far from the mean)
- Predicting $\bar{y}_R$ will have large errors

1.4 Finding the Best Split

Just like classification trees, we search for the best split by:

1. For each feature p
2. For each possible threshold t
3. Calculate the **weighted average MSE** of the two resulting regions:

$$\text{Split MSE} = \frac{N_1}{N} \cdot \text{MSE}(R_1) + \frac{N_2}{N} \cdot \text{MSE}(R_2) \tag{2}$$

4. Choose (p^*, t^*) that minimizes this weighted MSE

Example:

Simple Regression Tree Split Consider 1D data with X values from 0 to 10 and continuous Y values.

Try split at $X = 6.5$:

- Region 1 ($X \leq 6.5$): Contains samples with $\bar{y}_{R_1} = -0.008$
- Region 2 ($X > 6.5$): Contains samples with $\bar{y}_{R_2} = 0.697$

Calculate MSE for each region and their weighted average.

The algorithm tries **every possible split point** (every unique X value) and picks the one with lowest weighted MSE.

Warning

Computational Cost:

For each predictor with n unique values, we try n possible splits. With p predictors, that's potentially $n \times p$ calculations at each node.

For 1000 data points and 5 predictors: 5000 split evaluations per node!

1.5 Making Predictions

Once the tree is built:

1. A new data point traverses the tree (same as classification)

2. It lands in some leaf node
3. The prediction is the **mean** of training samples in that leaf

Example:

Regression Tree Prediction A trained tree has leaf node L_3 containing training samples with y values: $\{2.1, 2.3, 2.5, 2.0, 2.4\}$
The stored prediction for L_3 is: $\bar{y}_{L_3} = \frac{2.1+2.3+2.5+2.0+2.4}{5} = 2.26$
Any new sample that reaches L_3 gets prediction $\hat{y} = 2.26$

1.6 Visualizing Regression Trees

In 1D (one predictor):

- The predicted function is a **step function**
- Each step corresponds to a leaf node's mean
- More splits = more steps = more complex function

In 2D (two predictors):

- Feature space is partitioned into rectangles
- Each rectangle has a constant predicted value (the mean)
- It's like a "terraced landscape" with flat regions at different heights

1.7 Stopping Conditions

The same stopping conditions from classification trees apply:

- `max_depth`: Maximum tree depth
- `min_samples_leaf`: Minimum samples required in a leaf
- `max_leaf_nodes`: Maximum number of leaf nodes
- **MSE Gain Threshold**: Stop if MSE reduction from split is below threshold

Definition:

Accuracy Gain (MSE Reduction) The "accuracy gain" or MSE reduction from a split is:

$$\text{Gain} = \text{MSE}(\text{parent}) - \left[\frac{N_1}{N} \cdot \text{MSE}(R_1) + \frac{N_2}{N} \cdot \text{MSE}(R_2)\right] \tag{3}$$

If Gain $<$ threshold, don't split (the improvement isn't worth the added complexity).

1.8 Cross-Validation for Regression Trees

How do we choose the right stopping condition values?

Cross-validation! But with a different metric:

- Classification: Use accuracy, F1-score, or AUC
- Regression: Use MSE (or R^2) on the validation set

Key Information

MSE vs R^2 for Model Selection:

The instructor prefers using MSE over R^2 for cross-validation because:

- MSE is consistent across validation folds
- R^2 can vary because the mean $\bar{y}$ changes with different validation sets
- This introduces extra randomness in R^2 comparisons

2 Handling Categorical Variables

2.1 The Problem

Decision trees make splits based on comparisons: "Is $X > t$?"

For **numerical** features, this makes sense: "Is height > 178 cm?"

For **categorical** features, it doesn't: "Is color $>$ Red?" has no meaning!

2.2 Bad Approach: Ordinal Encoding

You might think: "Just assign numbers! Yellow=0, Red=1, Purple=2"

Warning

Why Ordinal Encoding Fails:
This creates an **artificial ordering** that doesn't exist in the data.
With Yellow=0, Red=1, Purple=2:

- Split at "Color ≥ 1" separates {Yellow} from {Red, Purple}

With Yellow=2, Red=0, Purple=1:

- Split at "Color ≥ 1" separates {Red} from {Yellow, Purple}

The tree structure depends on an arbitrary encoding choice!

2.3 Good Approach: One-Hot Encoding

Definition:

One-Hot Encoding (OHE) Convert each categorical variable into multiple binary (0/1) columns—one for each category.
Original: `Color` $\in$ {Yellow, Red, Purple}
After OHE:

- `Color_Yellow`: 1 if Yellow, 0 otherwise
- `Color_Red`: 1 if Red, 0 otherwise
- `Color_Purple`: 1 if Purple, 0 otherwise

Now splits make sense: "Is Color_Red ≥ 1?" means "Is the color Red?"

Example:

One-Hot Encoding Example **Before:**

Sepal Width	Color
3.0	Yellow
3.5	Red
3.7	Purple

After One-Hot Encoding:

Sepal Width	Color_Yellow	Color_Red	Color_Purple
3.0	1	0	0
3.5	0	1	0
3.7	0	0	1

Warning

Sklearn Warning:

Scikit-learn's `DecisionTreeClassifier` and `DecisionTreeRegressor` do **NOT** automatically handle categorical variables.

You must apply one-hot encoding **before** fitting the model:

```
import pandas as pd

# Method 1: Pandas get_dummies
X_encoded = pd.get_dummies(X, columns=['Color'])

# Method 2: sklearn OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse=False)
```

Note: Libraries like XGBoost, LightGBM, and CatBoost can handle categoricals natively.

Key Information

Dropping One Column:

With 3 categories, you only need 2 binary columns (the third is determined).

If Color_Yellow=0 and Color_Red=0, then it must be Purple.

However, for trees this redundancy usually doesn't matter much.

3 Pruning: A Better Way to Prevent Overfitting

3.1 The Problem with Stopping Conditions

We've seen stopping conditions like `max_depth`. But there's a problem:

- **Too restrictive:** Stop too early → underfitting
- **Too lenient:** Stop too late → overfitting
- **Hard to know in advance:** The optimal stopping point depends on the data

3.2 The Pruning Philosophy

Definition:

Pruning **Pruning** is a post-hoc technique where we:

1. First, grow the tree **fully** (or very deep)
2. Then, **remove** branches that don't improve performance

It's like letting the tree grow wild, then trimming it back carefully.

Key Information

Analogy: Pruning Real Trees

When pruning a real tree:

- Let it grow
- Cut back "dry, dead, or diseased" branches
- Result: A healthier, better-shaped tree

When pruning a decision tree:

- Let it grow (overfit)
- Cut back "useless" branches (that don't help on validation data)
- Result: A simpler, better-generalizing model

3.3 Cost Complexity Pruning (CCP)

The standard approach to pruning is **Cost Complexity Pruning**, which adds a penalty for tree size.

Definition:

Cost Complexity The **cost complexity** of a tree T is:

$$C_\alpha(T) = \text{Error}(T) + \alpha \cdot |T| \tag{4}$$

Where:

- $\text{Error}(T)$ = Classification error (or MSE for regression) on training data
- $|T|$ = Number of **leaf nodes** (terminal nodes)
- α = Complexity parameter (hyperparameter, $\alpha \geq 0$)

This is **regularization** applied to trees!

- Like Ridge regression adds $\lambda \sum \beta_j^2$ to penalize large coefficients
- CCP adds $\alpha \cdot |T|$ to penalize large trees

3.4 The Role of Alpha (α)

The complexity parameter α controls the trade-off:

- $\alpha = 0$**:** No penalty for tree size $\rightarrow$ favor large trees (full tree)
- $\alpha \rightarrow \infty$**:** Huge penalty $\rightarrow$ favor tiny trees (just the root)
- $0 < \alpha < \infty$**:** Balance between error and simplicity

Example:

Cost Complexity Comparison Let $\alpha = 0.2$

Full Tree T:

- $\text{Error}(T) = 0.32$
- $|T| = 8$ leaves
- $C_\alpha(T) = 0.32 + 0.2 \times 8 = \mathbf{1.92}$

Pruned Tree T':

- $\text{Error}(T') = 0.33$ (slightly worse training error)
- $|T'| = 7$ leaves
- $C_\alpha(T') = 0.33 + 0.2 \times 7 = \mathbf{1.73}$

Even though T' has higher training error, it has **lower cost complexity**!

The regularization favors T' because the added complexity of T isn't worth the small error reduction.

3.5 The Pruning Algorithm

Step 1: Generate Candidate Trees

Start with the full tree T_0. Iteratively remove the "weakest link"—the subtree whose removal **increases cost complexity the least** (or decreases it most).

This generates a sequence: $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_L$ (root only)

Step 2: Find Optimal Tree for Given α

For a fixed α, calculate $C_\alpha(T)$ for each candidate tree. Select the one with minimum cost complexity.

Step 3: Find Optimal α via Cross-Validation

Try different α values and use cross-validation to find which gives best validation performance.

Key Information

Nested Cross-Validation:

This is a two-level process:

1. **Inner loop:** For fixed α, find best pruned tree

2. **Outer loop:** Try different α values, select best via CV
This is computationally expensive but gives more robust model selection.

3.6 Pruning in Practice

```python
from sklearn.tree import DecisionTreeRegressor

# Train a full tree
tree = DecisionTreeRegressor(random_state=42)
tree.fit(X_train, y_train)

# Get the cost complexity pruning path
path = tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas  # Array of alpha values

# Train trees for each alpha value
trees = []
for alpha in ccp_alphas:
    t = DecisionTreeRegressor(ccp_alpha=alpha, random_state=42)
    t.fit(X_train, y_train)
    trees.append(t)

# Evaluate on validation set to choose best alpha
val_scores = [t.score(X_val, y_val) for t in trees]
best_alpha = ccp_alphas[np.argmax(val_scores)]

# Final model
final_tree = DecisionTreeRegressor(ccp_alpha=best_alpha)
final_tree.fit(X_train, y_train)
```

4 Important Details and Edge Cases

4.1 Can the Same Feature Be Used Multiple Times?

Yes! A feature can appear in multiple splits throughout the tree.

Example:

Multiple Splits on Same Feature Consider predicting income based on work experience:

- Root: "Work experience > 2 years?"
- Left child: "Work experience > 0.5 years?"
- Right child: "Work experience > 5 years?"

The same feature (work experience) is split at different thresholds at different levels. This allows the tree to capture non-linear relationships!

4.2 What Metrics for Cross-Validation?

For Classification:

- Accuracy (for balanced data)
- F1-score (for imbalanced data)
- AUC-ROC (for general performance)
- Custom utility function (if you care more about certain errors)

For Regression:

- MSE (preferred—consistent across folds)
- R^2 (works but less stable)
- MAE (if you want robustness to outliers)

4.3 Decision Boundary Shape

Important:

Linear vs Non-Linear Boundaries **max_depth=1:** Only one split → **Linear** decision boundary (single line/hyperplane)
max_depth≥2: Multiple splits → **Non-linear** boundaries (staircase pattern)
The deeper the tree, the more complex (more "wiggly") the boundary.

4.4 Total MSE of a Regression Tree

The overall MSE of a tree with multiple leaves is the **weighted average** of leaf MSEs:

$$\text{Total MSE} = \sum_{l=1}^{L} \frac{N_l}{N} \cdot \text{MSE}(R_l) \tag{5}$$

Example:

Computing Total MSE A tree has 4 leaf nodes:

- R_1: 90 samples, MSE = 0.2
- R_2: 5 samples, MSE = 1.2
- R_3: 3 samples, MSE = 1.5
- R_4: 2 samples, MSE = 1.8

Total samples: $N = 90 + 5 + 3 + 2 = 100$

Total MSE $= \frac{90}{100}(0.2) + \frac{5}{100}(1.2) + \frac{3}{100}(1.5) + \frac{2}{100}(1.8)$

$= 0.18 + 0.06 + 0.045 + 0.036 = \mathbf{0.321}$

Note: The total is dominated by R_1 because it contains 90% of samples!

5 Summary and Key Takeaways

Key Summary

Regression Trees:

- Same structure as classification trees, different criterion
- Splitting criterion: Minimize **weighted average MSE**
- Prediction: **Mean** of training samples in leaf node
- Stopping conditions: Same as classification (max_depth, min_samples_leaf, etc.)

Categorical Variables:

- Cannot use ordinal encoding (creates artificial order)
- Use **One-Hot Encoding** to create binary columns
- Sklearn requires manual OHE before fitting

Pruning:

- Alternative to pre-defined stopping conditions
- "Grow full, then cut back"
- Cost complexity: $C_\alpha(T) = \text{Error}(T) + \alpha|T|$
- α controls trade-off between error and simplicity
- Find optimal α via cross-validation

6 Learning Checklist

☐ Can you explain the difference between classification and regression trees?

☐ Do you understand why MSE is used as the splitting criterion for regression?

☐ Can you explain what the prediction at a leaf node represents in regression trees?

☐ Do you know why ordinal encoding is problematic for categorical variables?

☐ Can you apply one-hot encoding to categorical features?

☐ Do you understand the cost complexity formula: $C_\alpha(T) = \text{Error}(T) + \alpha|T|$?

☐ Can you explain how α affects the complexity of the pruned tree?

☐ Do you know the two-level cross-validation process for finding optimal α?

☐ Can you compute the total MSE of a tree from its leaf node MSEs?

☐ Do you understand why `max_depth=1` gives a linear decision boundary?

7 Looking Ahead

Next lectures will cover **ensemble methods**:

- **Bagging:** Building many trees on bootstrap samples
- **Random Forests:** Bagging + random feature selection
- **Boosting:** Building trees sequentially to correct errors

- **Gradient Boosting:** XGBoost, LightGBM—state-of-the-art methods

These methods combine multiple trees to create more powerful and robust models!

Lecture 20: Missing Data and Model Visualization

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 20
- **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb
- **Topics:** Missing Data, MCAR/MAR/MNAR, Imputation Methods, Model Visualization, Partial Dependence Plots

Lecture Overview

This lecture covers two important practical topics in data science:

1. **Missing Data:** Real datasets almost always have missing values. Understanding *why* data is missing and handling it properly is crucial to avoid biased models.
2. **Model Visualization:** For "black-box" models like KNN and decision trees, we can't simply look at coefficients. Partial Dependence Plots (PDPs) help us understand how features affect predictions.

Key Insight: Both topics are about dealing with the messiness of real-world data science—incomplete data and complex models that resist simple interpretation.

Contents

1 What is Missing Data?

1.1 The Problem

Definition:

Missing Data **Missing data** (or **missingness**) refers to values that should exist in a dataset but are absent. In Python/Pandas, these appear as `NaN` (Not a Number) or `NA`.

Missing data creates two problems:

1. **Technical:** Most ML libraries (including sklearn) throw errors when they encounter NaN values
2. **Statistical:** Improper handling can introduce **bias** into predictions and estimates

Warning

What is Bias?

Bias means your estimates or predictions are **systematically wrong**—not just randomly off, but consistently wrong in a particular direction.

Analogy: A scale that always reads 2 lbs too high is biased. No matter how many times you weigh yourself, you'll always be 2 lbs over your true weight.

In modeling:

- **Biased prediction:** Your $\hat{y}$ systematically over- or under-estimates the true y
- **Biased estimation:** Your $\hat{\beta}$ systematically differs from the true β

1.2 Why Does Missing Data Occur?

Missing data can arise for many reasons:

- **Data entry errors:** Someone accidentally skipped a field
- **Non-response:** Survey respondent chose not to answer a question
- **New variable collection:** You started measuring something new midway through data collection
- **Measurement limits:** Sensor couldn't detect values below a threshold
- **Study dropout:** Participant left a clinical trial

Important:

The Million-Dollar Question **Why** is the data missing? The answer determines:

- Whether simple approaches will work
- How much bias you might introduce
- What imputation strategy to use

The reason for missingness is the key to proper handling.

1.3 Simple (But Problematic) Approaches

Before taking this class, you might have done:

Option 1: Drop rows with missing values

```
df_clean = df.dropna()  # Remove any row with NaN
```

Option 2: Fill with mean/median/mode

```
df['X'].fillna(df['X'].mean(), inplace=True)
```

Warning

Problems with Simple Approaches:

Dropping rows:

- Loses valuable data
- If missingness is related to outcome, introduces bias
- If 30% of rows have missing values, you lose 30% of data!

Mean imputation:

- Artificially reduces variance (all missing values become the same)
- Weakens relationships between variables
- Doesn't account for uncertainty in imputed values

2 Types of Missing Data: MCAR, MAR, MNAR

Understanding *why* data is missing is crucial. Statisticians classify missing data into three types:

2.1 MCAR: Missing Completely at Random

Definition:

MCAR Data is **Missing Completely at Random** if the probability of missingness is unrelated to:

- The missing value itself
- Any other observed variable

Missingness is purely random—like someone randomly punched holes in your spreadsheet.

Example: While entering survey data into Excel, the data entry person randomly skipped some values due to fatigue. The skipping had nothing to do with what the values were or who the respondents were.

Good news: With MCAR, dropping rows or simple imputation won't introduce bias (though you may lose statistical power).

Bad news: True MCAR is rare in practice.

2.2 MAR: Missing at Random

Definition:

MAR Data is **Missing at Random** if the probability of missingness can be fully explained by **other observed variables**—but not by the missing value itself.

Example: In a survey about workplace harassment, people may choose not to answer based on their gender (which you observed), not based on whether they were actually harassed.

Key point: "Random" here is misleading. It means "random *conditional on observed data*." If you know gender, the missingness becomes random.

Good news: With MAR, we can use other observed variables to model and correct for missingness.

2.3 MNAR: Missing Not at Random

Definition:

MNAR Data is **Missing Not at Random** if the probability of missingness depends on:

- The unobserved (missing) value itself, OR
- Some unobserved variable

Example 1: High-income people are less likely to report their income *because* it's high. The missing value itself affects missingness.

Example 2: Patients drop out of a clinical trial because they experienced severe side effects. The side effect severity (unmeasured) causes both dropout and likely worse outcomes.

Bad news: MNAR is the hardest case. No statistical method can fully correct for bias because the information needed is... missing!

Table 1: *Summary of Missing Data Types*

Type	Missingness Depends On	Example	Handling Difficulty
MCAR	Nothing (pure random)	Data entry error	Easiest
MAR	Observed variables only	Gender affects response rate	Moderate
MNAR	Missing value or unobserved variables	High earners hide income	Hardest

Important:

The Fundamental Problem **Can you ever know which type you have?**

No. You cannot definitively determine from the data alone whether missingness is MCAR, MAR, or MNAR. MNAR involves unmeasured variables—by definition, you can't see them.

Practical approach: Assume MAR and use the best imputation methods available. This at least handles the recoverable bias.

3 Imputation Methods

3.1 Important Considerations

Before choosing an imputation method, consider:

1. **Where is the missingness?**
   - In predictors (X): Most imputation methods handle this
   - In response (Y): Much harder—often just drop these rows
2. **Variable type:** Numeric vs categorical (different methods apply)
3. **Amount of missingness:**
   - $< 10\%$: Usually safe to impute
   - $10 - 50\%$: Be careful; imputation may add noise
   - $> 50\%$: Consider dropping the variable entirely

3.2 Method 1: Missingness Indicator Variable

This simple method treats "missing" as **information itself**:

1. For variable X_1 with missing values:
2. Create X_1^*: Impute missing values with 0 (or mean)
3. Create X_1^{miss}: Binary indicator (1 if was missing, 0 otherwise)
4. Use **both** X_1^* and X_1^{miss} in your model

Example:

Missingness Indicator

Original		**Imputed**		**Indicators**	
X1	X2	X1*	X2*	X1_miss	X2_miss
10	0	10	0	0	0
5	1	5	1	0	0
21	NaN	21	**0**	0	**1**
NaN	0	**0**	0	**1**	0

Now the model can learn: "When X2_miss=1, treat X2*=0 differently than when X2_miss=0"

Key Information

Why This Works:

This creates a "did not respond" category. The model can estimate separate effects for:

- Observed zeros
- Imputed zeros (actually missing)

This is especially useful when MNAR is suspected—the fact that someone didn't respond might itself be predictive!

3.3 Method 2: Model-Based Imputation

Treat imputation as a **prediction problem**:

1. Identify variable X_j with missing values
2. **Training set:** Rows where X_j is observed
3. **Test set:** Rows where X_j is missing
4. Fit a model: $X_j \sim X_1, X_2, ..., X_{j-1}, X_{j+1}, ..., X_p$
5. Predict $\hat{X}_j$ for the test set
6. Fill in the missing values with $\hat{X}_j$

Models you can use:

- **Linear regression** (for numeric X_j)
- **Logistic regression** (for binary X_j)
- **KNN** (for any type)
- **Decision trees** (for any type)

Example:

KNN Imputation Variable Y (numeric) has missing values. Variable X (color) is fully observed. Using KNN with K=2:
For a missing value at color = "medium red":

- Find 2 nearest neighbors: "dark red" (Y=1) and "light red" (Y=0.5)
- Impute: $\hat{Y} = \frac{1+0.5}{2} = 0.75$

For a missing value at color = "yellow":

- Find 2 nearest neighbors: "orange" (Y=0.1) and "green" (Y=10)
- Impute: $\hat{Y} = \frac{0.1+10}{2} = 5.05$

3.4 Method 3: Imputation with Uncertainty

The problem with Method 2: **imputed values are too perfect**.

If you use linear regression to impute, all imputed values fall exactly on the regression line. But real data has scatter!

Warning

The Problem with Deterministic Imputation:
If you impute with $\hat{y}$ (the prediction), your imputed values have **no variance around the prediction**. This:

- Artificially strengthens relationships
- Makes your model overconfident
- Underestimates uncertainty in downstream analyses

Solution: Add randomness!

For linear regression:

1. Predict $\hat{y}$
2. Randomly sample a residual ϵ from training data (or from $N(0, \hat{\sigma}^2)$)
3. Impute: $\hat{y} + \epsilon$

For KNN:

1. Find K neighbors
2. Instead of averaging, **randomly sample one** of the K values
3. Impute that sampled value

For decision trees:

1. Traverse to the leaf node
2. Instead of using the leaf mean, **randomly sample one** observation from that leaf
3. Impute that sampled value

For classification:

1. Get predicted probability $\hat{p}$
2. **Flip a coin** with probability $\hat{p}$ to determine class
3. Don't just use the majority class!

3.5 Method 4: Iterative Imputation (Multiple Variables)

What if **multiple** variables have missing values?

You can't impute X_1 from X_2, X_3 if X_2 and X_3 also have missing values!

Solution: Iterate!

1. Initialize: Fill all missing values with simple imputation (mean/median)
2. Round 1: Use current X_2, X_3 to impute X_1
3. Round 1: Use current X_1, X_3 to impute X_2
4. Round 1: Use current X_1, X_2 to impute X_3
5. Repeat rounds until values converge (stop changing significantly)

```
from sklearn.impute import IterativeImputer

# Iteratively imputes using all other features
imputer = IterativeImputer(max_iter=10, random_state=42)
X_imputed = imputer.fit_transform(X)
```

3.6 Sklearn Imputation Tools

```
from sklearn.impute import SimpleImputer, KNNImputer, IterativeImputer
from sklearn.impute import MissingIndicator

# 1. Simple imputation (mean, median, most_frequent, constant)
```

```
simple = SimpleImputer(strategy='mean')

# 2. KNN imputation
knn = KNNImputer(n_neighbors=5)

# 3. Iterative imputation (most sophisticated)
iterative = IterativeImputer(max_iter=10, random_state=0)

# 4. Create missingness indicators
indicator = MissingIndicator()

# Usage
X_imputed = simple.fit_transform(X)
X_indicator = indicator.fit_transform(X)
```

4 Visualizing Black-Box Models

4.1 The Interpretation Problem

For **parametric models** (linear/logistic regression):

- We have coefficients β_j
- Interpretation: "One unit increase in X_j leads to β_j change in Y"
- These are "white-box" models—we can see inside

For **non-parametric models** (KNN, decision trees, random forests):

- No simple coefficients to interpret
- Complex decision rules or distance calculations
- These are "black-box" models—hard to see inside

Key Information

Why Do We Need Interpretation?
Even with black-box models, we need to:

- Understand what the model learned
- Check if relationships make domain sense
- Communicate findings to stakeholders
- Debug unexpected behavior

Solution: Plot the predictions!

4.2 Partial Dependence Plots (PDPs)

Definition:

Partial Dependence Plot A **Partial Dependence Plot (PDP)** shows the relationship between a feature X_j and the predicted outcome $\hat{Y}$, while holding all other features at fixed values (typically their medians).
It answers: "How does changing X_j affect predictions, controlling for everything else?"

How to create a PDP:

1. Choose the feature of interest, X_j
2. Fix all other features at their median values
3. Create a sequence of X_j values: $X_j = \{x_1, x_2, ..., x_n\}$
4. For each x_i, predict $\hat{Y}$ using the model
5. Plot X_j vs $\hat{Y}$

Example:

PDP for Heart Disease Prediction Model: KNN (K=50) predicting heart disease from max heart rate and other variables.

Create PDP for max heart rate:

1. Fix Age, Sex, RestBP, etc. at their medians
2. Vary MaxHR from 70 to 200
3. For each MaxHR value, predict probability of heart disease
4. Plot MaxHR vs predicted probability

Result: A curve showing that higher max heart rate → lower probability of heart disease (negative relationship).

```python
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

# Fit model
knn = KNeighborsClassifier(n_neighbors=50)
knn.fit(X_train, y_train)

# Create synthetic X values for max heart rate
maxhr_range = np.linspace(X['MaxHR'].min(), X['MaxHR'].max(), 100)

# Hold other variables at their medians
other_medians = X.drop('MaxHR', axis=1).median()

# Create prediction data
pred_data = pd.DataFrame({
    'MaxHR': maxhr_range,
    **{col: [val]*100 for col, val in other_medians.items()}
})

# Predict probabilities
probs = knn.predict_proba(pred_data)[:, 1]

# Plot PDP
import matplotlib.pyplot as plt
plt.plot(maxhr_range, probs)
plt.xlabel('Max Heart Rate')
plt.ylabel('Predicted P(Heart Disease)')
plt.title('Partial Dependence Plot')
```

4.3 Using PDPs to Detect Interactions

The power of PDPs comes from comparing them across different subgroups:

Key insight: If the PDP shape changes when you vary another feature, there's an **interaction**.

Example:

Detecting Age x MaxHR Interaction Create three PDPs for MaxHR:

1. Age = minimum (29 years)
2. Age = median (55 years)
3. Age = maximum (77 years)

If the three curves are different, it means the effect of MaxHR on heart disease depends on Age—an interaction!

Typical finding:

- At low MaxHR: All ages have high disease probability
- At high MaxHR: Young people benefit more (lower probability) than older people

Interpretation: High max heart rate (fitness) is more protective for younger patients.

Warning

Limitations of PDPs:

- Assumes features are independent (can be misleading if features are correlated)
- Shows average effect; may hide heterogeneity
- Doesn't show uncertainty in the relationship

For more sophisticated interpretation, consider SHAP values or ICE plots.

5 Summary and Key Takeaways

Key Summary

Missing Data:

- Missing data causes both technical problems (sklearn errors) and statistical problems (bias)
- Three types: MCAR (random), MAR (explained by observed data), MNAR (depends on missing value itself)
- You can never know for sure which type you have
- Simple approaches (drop rows, mean imputation) can introduce bias

Imputation Methods:

- **Missingness indicator:** Create binary flag for "was missing"
- **Model-based:** Predict missing values from other features
- **With uncertainty:** Add randomness to imputed values
- **Iterative:** For multiple variables with missingness

Model Visualization:

- Black-box models (KNN, trees) need visualization for interpretation
- **Partial Dependence Plots:** Show feature effect while holding others constant
- Compare PDPs across subgroups to detect interactions

6 Learning Checklist

☐ Can you explain what bias means in the context of estimation and prediction?

☐ Do you understand the three types of missing data (MCAR, MAR, MNAR)?

☐ Can you explain why you can never definitively know which type of missingness you have?

☐ Do you understand the problems with simple approaches (dropping rows, mean imputation)?

☐ Can you implement the missingness indicator variable approach?

☐ Do you understand model-based imputation and how to add uncertainty?

☐ Can you explain iterative imputation for multiple variables?

☐ Do you understand what a Partial Dependence Plot shows?

☐ Can you use PDPs to detect interactions between features?

7 Looking Ahead

Next lectures cover **ensemble methods**:

- **Bagging:** Bootstrap aggregating—averaging many trees
- **Random Forests:** Bagging + random feature selection
- **Boosting:** Sequential models that correct previous errors
- **Gradient Boosting:** XGBoost, LightGBM—top performers on tabular data

■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 21: Bagging and Out-of-Bag Error

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Objective:** Understand ensemble learning through Bagging and learn how to use Out-of-Bag error for model validation

Contents

1 The Big Picture: Why We Need Bagging

Key Summary

This lecture introduces **Bagging** (**B**ootstrap **Agg**regat**ing**), a powerful technique that combines multiple decision trees to create a stronger, more reliable model. We'll also learn about **Out-of-Bag (OOB) error**, a clever way to evaluate our model without needing a separate validation set.

1.1 Where We Are in the Course

Let's recap what we've learned so far about decision trees:

1. **Classification trees**: Split data to maximize purity (using Gini index or entropy)
2. **Regression trees**: Split data to minimize MSE within each region
3. **Controlling complexity**: We use max depth, minimum samples per leaf, or pruning
4. **Cross-validation**: We use it to choose the best hyperparameters

But here's the problem: **single decision trees have limitations**.

1.2 The Fundamental Problem with Single Trees

Decision trees make **axis-aligned splits**—they can only draw vertical and horizontal lines to separate data. This means:

- If the true decision boundary is complex (like a circle), you need **many splits** to approximate it
- Many splits means a **deep tree**
- Deep trees tend to **overfit**

Example: Approximating a Circular Boundary

Imagine your data has two classes separated by a circular boundary. With decision trees:

- **Depth 1-2**: Makes a few horizontal/vertical cuts. Result: **severe underfitting**—the splits can't capture the circle at all
- **Depth 4-5**: Starts to form a rough rectangular approximation of the circle
- **Depth 20+**: Creates many small rectangles that start to look like a circle, but now you're **overfitting** to noise in your training data

The dilemma: go shallow and underfit, or go deep and overfit. There's no perfect depth!

Warning

The Bias-Variance Tradeoff in Trees:

- **Shallow trees**: High bias, low variance (underfit)
- **Deep trees**: Low bias, high variance (overfit)

Single trees struggle to achieve **both** low bias AND low variance.

1.3 The Key Insight: Multiple Opinions Are Better Than One

Think about how you make important decisions in real life:

Example: The Medical Second Opinion Analogy

Suppose you get an MRI scan and one doctor tells you that you need surgery. What do you do? Most people get a **second opinion**. Maybe even a third. Why?

- Each doctor might make mistakes (high variance)
- Different doctors were trained at different hospitals with different experiences
- By combining multiple opinions, individual errors tend to **cancel out**

If 3 doctors say "surgery needed" and 1 says "no surgery," you'd probably go with the majority. **This is exactly the idea behind ensemble learning!**

1.4 The Mathematical Insight: Averaging Reduces Variance

Here's a beautiful statistical fact that underlies all of ensemble learning:

Definition: The Power of Averaging

If you have n independent random variables, each with variance σ^2, then the variance of their average is:

$$\text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{\sigma^2}{n}$$

In plain English: **averaging reduces variance by a factor of** n.

This is a profound insight! It means:

- One tree might overfit and give a wrong prediction
- But if we train many trees and average their predictions, the errors wash out
- The key requirement: the trees need to be **somewhat independent** (make different errors)

Key Information

Historical Note: This idea traces back to Carl Friedrich Gauss in the late 1700s. When astronomers were trying to calculate planetary orbits, they initially thought they needed exactly 6 measurements for 6 unknowns. Gauss showed that **more measurements actually reduce error**—what seemed like "too much data" actually improved accuracy because random errors cancel out. This was one of the founding insights of statistics!

2 What is Bagging?

Definition: Bagging

Bagging stands for **B**ootstrap **Agg**regat**ing**. It's an ensemble method that:

1. Creates multiple different training datasets using **bootstrap sampling**
2. Trains a separate model (typically a deep decision tree) on each dataset
3. **Aggregates** their predictions (majority vote for classification, average for regression)

The name "Bagging" was coined by Leo Breiman in 1996 when he introduced this technique. It's a portmanteau of "Bootstrap" and "Aggregating."

2.1 Step 1: Bootstrap Sampling

The first challenge: we want to train multiple trees on **different** training data, but we only have **one** dataset. How do we create multiple datasets?

Definition: Bootstrap Sampling

Bootstrap sampling creates new datasets by sampling **with replacement** from the original data:

1. Start with original dataset of N samples
2. Randomly select N samples WITH REPLACEMENT (same sample can be picked multiple times)
3. This creates one "bootstrap sample"
4. Repeat to create as many bootstrap samples as you want

Example: Bootstrap in Action

Suppose your original dataset has 5 animals: {cow, cat, horse, dog, rat}
Three bootstrap samples might look like:

Sample	Contents
Bootstrap 1	dog, cat, dog, horse, horse
Bootstrap 2	rat, rat, cow, dog, cow
Bootstrap 3	dog, rat, horse, cat, rat

Notice:

- Bootstrap 1 has dog 3 times, horse 2 times, but NO rat or cow
- Bootstrap 2 has rat and cow twice each
- Each sample is the same SIZE as the original (5 animals), but with different compositions

Important: The 36.8% Rule

On average, each bootstrap sample will be missing about **36.8%** of the original data points. Why? The probability that a specific point is NOT chosen in any of the N draws is:

$$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} \approx 0.368$$

This means roughly one-third of the data is "out of bag" (OOB) for each tree. We'll use this fact later!

2.2 Step 2: Train Multiple Trees

Once we have B bootstrap samples, we train one decision tree on each sample:

- **Tree 1** is trained on Bootstrap Sample 1
- **Tree 2** is trained on Bootstrap Sample 2
- ... and so on
- **Tree B** is trained on Bootstrap Sample B

Key Information

In bagging, we typically let each tree grow **deep** (high complexity). Why? Because even though deep trees overfit, the averaging process will reduce this overfitting. We're intentionally using low-bias, high-variance models and then reducing variance through aggregation.

2.3 Step 3: Aggregate Predictions

When a new data point comes in, we get predictions from ALL trees and combine them:

For Classification:

$$\hat{y} = \text{majority vote of } \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_B\}$$

Each tree "votes" for a class, and we pick the class with the most votes.

For Regression:

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} \hat{y}_b$$

We simply average all the predictions.

Example: Complete Bagging Example

Task: Predict whether a patient has diabetes (classification)

Setup:

- Original training data: 1000 patients
- Number of trees: $B = 100$
- Tree depth: 20 (deep trees)

Training:

1. Create 100 bootstrap samples, each with 1000 patients (with replacement)
2. Train a depth-20 decision tree on each sample

Prediction for a new patient:

1. Send patient through all 100 trees
2. Suppose 73 trees predict "Diabetes" and 27 predict "No Diabetes"
3. Final prediction: "Diabetes" (majority vote)

The confidence might even be interpreted as 73%, giving us a probability-like output!

3 Why Bagging Works

3.1 Visualizing the Magic

Let's see how bagging reduces variance visually:

Example: Decision Boundaries at Different Depths

Consider a classification problem with a complex decision boundary.

With Single Trees:

- **Depth 2**: Decision boundary is too simple (underfit)
- **Depth 5**: Starting to capture the shape
- **Depth 100**: Captures detail but is extremely "wiggly" (overfit)

With Bagging (100 trees, depth 100):

- Each individual tree has a wiggly, overfit boundary
- But when we average/vote, the individual wiggles "cancel out"
- The average boundary is much smoother while still capturing the true pattern

It's like taking a photo with a shaky camera: one shot is blurry, but averaging 100 shots gives you a clearer image.

3.2 The Key Advantages of Bagging

1. **High Expressiveness**: Each tree can be deep, capturing complex patterns
2. **Low Variance**: Averaging multiple predictions reduces overall variance
3. **Reduced Overfitting**: Individual trees overfit differently; averaging cancels out
4. **Works with Any Base Model**: Though trees are most common, bagging works with any algorithm

Key Information

Bagging as Regularization:

Regularization is any technique that prevents overfitting. Usually we think of it as penalizing model complexity (like L1/L2 penalties). But bagging achieves regularization through a different mechanism: **averaging out the variance**. This makes bagging a form of "implicit regularization."

3.3 Can We Use Bagging with Other Models?

Yes! While bagging is most commonly used with decision trees, the concept applies to any model:

- **Bagged Linear Regression**: Train linear regression on bootstrap samples, average predictions
- **Bagged Neural Networks**: Train neural networks on bootstrap samples, average predictions
- **Bagged Any-Model**: Same principle applies

However, in practice:

- Bagging helps most when the base model has **high variance** (like deep trees)
- For already-stable models (like linear regression), the benefit is smaller

- For expensive models (like deep neural networks), the computational cost is prohibitive

4 Hyperparameters in Bagging

Bagging introduces new hyperparameters on top of the single-tree hyperparameters.

4.1 Single Tree Hyperparameters (Still Apply)

These control the complexity of each individual tree:

- **max_depth**: Maximum depth of each tree
- **min_samples_split**: Minimum samples needed to split a node
- **min_samples_leaf**: Minimum samples required in each leaf
- **criterion**: Gini, entropy (classification) or MSE (regression)

4.2 New Bagging Hyperparameter: Number of Estimators

The most important new hyperparameter is **n_estimators**—the number of trees in the ensemble.

Definition: Number of Estimators

The **number of estimators** (B or `n_estimators` in sklearn) is the number of trees in the bagging ensemble.
Key property: **More trees always helps (up to a point)** and **never hurts**!

Important: n_estimators Does NOT Control Complexity

This is a crucial distinction:

- **max_depth**: Controls bias-variance tradeoff (too much = overfit)
- **n_estimators**: Controls variance only (more = less variance)

Unlike max_depth, increasing n_estimators will NOT cause overfitting. It only reduces variance.

Example: How Error Changes with n_estimators

Training Error: As n_estimators increases, training error **goes up slightly**.
Why? With one deep tree, you can perfectly memorize the training data. But when you average many trees, some of that memorization washes out.
Validation Error: As n_estimators increases, validation error **goes down significantly**.
Why? The overfitting from individual trees cancels out, so the model generalizes better.
The validation error will decrease and then **flatten out**—it won't go back up!

4.3 The Hyperparameter Explosion Problem

With bagging, we need to tune:

- Splitting criterion (Gini vs. Entropy vs. MSE)
- Stopping condition (depth, min samples, etc.)
- Values for each stopping condition
- Number of estimators

If each has 5-10 options, we might have $3 \times 4 \times 10 \times 10 = 1,200$ combinations!

With 5-fold cross-validation, that's $1,200 \times 5 = 6,000$ models to train.

This is getting expensive! We need a better way to validate...

5 Out-of-Bag (OOB) Error

Key Summary

Out-of-Bag (OOB) error is a clever technique that uses the bootstrap's "waste" as a free validation set. It lets us evaluate our model without setting aside data for validation or performing expensive cross-validation.

5.1 The Key Insight

Remember: each bootstrap sample leaves out about 36.8% of the original data. This "left out" data is called the **Out-of-Bag (OOB)** data for that tree.

Definition: Out-of-Bag Data

For tree b, the **Out-of-Bag samples** are all data points from the original training set that were NOT included in bootstrap sample b.
Since the tree never saw these points during training, they act like a built-in validation set!

Example: OOB Samples Illustration

Original data: $\{x_1, x_2, x_3, x_4, x_5\}$

Tree	Bootstrap Sample	OOB Samples
Tree 1	$\{x_1, x_1, x_3, x_3, x_5\}$	$\{x_2, x_4\}$
Tree 2	$\{x_2, x_2, x_4, x_5, x_5\}$	$\{x_1, x_3\}$
Tree 3	$\{x_1, x_3, x_3, x_4, x_5\}$	$\{x_2\}$

Each tree has different OOB samples. We can use this to get a validation error for each data point!

5.2 Computing OOB Error

The OOB error calculation works as follows:

1. **For each data point x_i:**
   (a) Find all trees where x_i was OUT of the bootstrap sample
   (b) Get predictions from only those trees
   (c) Aggregate these predictions (majority vote or average)
   (d) Compare to the true label y_i
2. **Average the errors across all data points**

Definition: OOB Error Formula

For Classification:

Let $\hat{y}_i^{OOB}$ be the majority vote from trees that didn't see x_i:

$$\text{OOB Error} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{y}_i^{OOB} \neq y_i]$$

For Regression:
Let $\hat{y}_i^{OOB}$ be the average prediction from trees that didn't see x_i:

$$\text{OOB Error} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i^{OOB})^2$$

Example: Step-by-Step OOB Calculation

Using our earlier example with 5 data points and 3 trees:

Computing OOB prediction for x_2:

- Tree 1: x_2 is OOB, so use Tree 1's prediction
- Tree 2: x_2 is IN the bootstrap, so skip
- Tree 3: x_2 is OOB, so use Tree 3's prediction

If Tree 1 predicts "Class A" and Tree 3 predicts "Class B":

- For classification: It's a tie! Typically we'd use more trees to break ties.
- For regression: Average the two predictions

Repeat for all data points, then compute overall error.

5.3 Why OOB Error is Great

1. **It's FREE:** No extra computation—we're using data that was "wasted" anyway
2. **No data splitting needed:** We don't have to set aside a validation set, so we can use ALL data for training
3. **Lower leakage than CV:** In cross-validation, each data point appears in training sets for $(K-1)$ folds. There's subtle information leakage. OOB has **zero leakage** because each prediction uses only trees that never saw that point.
4. **Automatically computed:** Sklearn computes it for you with `oob_score=True`

Important: OOB is for Validation

OOB error replaces your **validation set**. Use it for:

- Hyperparameter tuning
- Model selection
- Comparing different configurations

But you still need a **separate test set** for final, unbiased evaluation!

The workflow:

1. Split data: 80% train, 20% test (test goes in a vault)
2. Train bagging model with `oob_score=True`

3. Use OOB error to tune hyperparameters
4. Once happy, evaluate ONCE on the test set

6 Bagging in Python with sklearn

6.1 Basic Implementation

```python
from sklearn.ensemble import BaggingClassifier, BaggingRegressor
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.model_selection import train_test_split

# Load your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create bagging classifier
bagging_clf = BaggingClassifier(
    estimator=DecisionTreeClassifier(max_depth=20),  # Base model
    n_estimators=100,      # Number of trees
    oob_score=True,        # Enable OOB scoring
    random_state=42,
    n_jobs=-1              # Use all CPU cores
)

# Train
bagging_clf.fit(X_train, y_train)

# Get OOB score (for validation)
print(f"OOB Accuracy: {bagging_clf.oob_score_:.4f}")

# Final test score
print(f"Test Accuracy: {bagging_clf.score(X_test, y_test):.4f}")
```

6.2 Key Parameters

Table 1: *Important BaggingClassifier/BaggingRegressor Parameters*

Parameter	Default	Description
`estimator`	None	Base estimator (uses DecisionTree if None)
`n_estimators`	10	Number of trees in ensemble
`max_samples`	1.0	Fraction of samples for each bootstrap
`max_features`	1.0	Fraction of features for each tree
`bootstrap`	True	Whether to use bootstrap sampling
`oob_score`	False	Whether to compute OOB error
`n_jobs`	None	Number of parallel jobs (-1 for all cores)

6.3 Tuning with OOB Error

```python
import numpy as np
import matplotlib.pyplot as plt

```

```python
# Test different numbers of estimators
n_estimators_range = [10, 25, 50, 100, 200, 500]
oob_scores = []
test_scores = []

for n in n_estimators_range:
    clf = BaggingClassifier(
        estimator=DecisionTreeClassifier(max_depth=15),
        n_estimators=n,
        oob_score=True,
        random_state=42
    )
    clf.fit(X_train, y_train)
    oob_scores.append(clf.oob_score_)
    test_scores.append(clf.score(X_test, y_test))

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, oob_scores, 'b-o', label='OOB Score')
plt.plot(n_estimators_range, test_scores, 'r-s', label='Test Score')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees')
plt.legend()
plt.grid(True)
plt.show()
```

6.4 Comparing with Single Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier

# Single deep tree
single_tree = DecisionTreeClassifier(max_depth=20)
single_tree.fit(X_train, y_train)
print(f"Single Tree - Train: {single_tree.score(X_train, y_train):.4f}")
print(f"Single Tree - Test:  {single_tree.score(X_test, y_test):.4f}")

# Bagged trees
bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(max_depth=20),
    n_estimators=100,
    oob_score=True,
    random_state=42
)
bagging.fit(X_train, y_train)
print(f"Bagging - Train: {bagging.score(X_train, y_train):.4f}")
print(f"Bagging - OOB:   {bagging.oob_score_:.4f}")
print(f"Bagging - Test:  {bagging.score(X_test, y_test):.4f}")

```

```
# You'll typically see:
# - Single tree: High training, low test (overfit)
# - Bagging: Similar or lower training, much higher test
```

7 Limitations of Bagging

Bagging is powerful, but it has some important limitations:

7.1 Loss of Interpretability

Warning

One of the biggest advantages of decision trees is **interpretability**—you can explain exactly why a prediction was made by following the tree.
With bagging, we have 100+ trees, each making slightly different decisions. There's no single "path" to explain. The interpretability is lost.
We'll learn about techniques like **feature importance** in the Random Forest lecture to partially address this.

7.2 Still Need to Control Depth

While bagging reduces variance, you can't just set `max_depth=1000` and expect magic:

- **Too shallow**: Even averaging many underfit trees gives an underfit result
- **Too deep**: Computational cost increases, and correlation between trees increases

You still need to tune the depth, just less aggressively than with single trees.

7.3 The Correlation Problem (MAJOR)

This is the most important limitation and motivates the next topic (Random Forests).

Important: Trees Are Often Correlated!

The mathematical formula for variance reduction assumes trees are **independent**. But in practice, they're often highly **correlated**.
Why? If one feature (e.g., "Glucose" for diabetes prediction) is extremely predictive:

- Tree 1 will split on Glucose at the root
- Tree 2 will split on Glucose at the root
- Tree 3 will split on Glucose at the root
- ... and so on

All trees end up looking very similar! They make the same errors, so averaging doesn't help as much.

Example: Visualizing Tree Correlation

Look at bagged trees for diabetes prediction:

	Tree 1	**Tree 2**	**Tree 3**
Root Node	Glucose	Glucose	Glucose
Level 2	BMI	Blood Pressure	BMI
Level 3	Age	Age	Blood Pressure

All three trees start with Glucose! This correlation limits how much variance we can reduce.

This limitation leads us directly to **Random Forests**, which we'll cover in the next lecture. Random Forests add a clever twist: when building each tree, they only consider a **random subset of features** at each split. This "decorrelates" the trees and makes bagging much more effective.

8 Key Takeaways

Key Summary

What We Learned Today:

1. **The Problem:** Single decision trees struggle with bias-variance tradeoff
2. **The Solution:** Ensemble learning—combine multiple models
3. **Bagging = Bootstrap + Aggregating:**
   - Create diverse training sets via bootstrap sampling
   - Train a model on each (typically deep decision trees)
   - Aggregate predictions (vote or average)
4. **Why It Works:** Averaging independent predictions reduces variance
5. **Key Hyperparameters:**
   - Tree depth (controls complexity/bias-variance)
   - Number of estimators (reduces variance, never causes overfit)
6. **OOB Error:** Free validation using bootstrap "leftovers"
   - About 36.8% of data is OOB for each tree
   - Replaces need for validation set
   - More robust than cross-validation
7. **Limitations:**
   - Loss of interpretability
   - Tree correlation limits effectiveness
8. **Coming Up:** Random Forests solve the correlation problem!

Table 2: *Comparison: Single Tree vs. Bagging*

Aspect	Single Tree	Bagging
Bias	Can be low (deep tree)	Can be low
Variance	High	Low (averaging)
Interpretability	High	Low
Computation	Fast	Slower (many trees)
Overfitting Risk	High	Lower

9 Practice Questions

1. **Why does averaging reduce variance?** Explain the mathematical principle behind why combining multiple predictions gives a more stable result.
2. **Bootstrap Sampling:** If you have 1000 data points and create a bootstrap sample of 1000 points, approximately how many unique points will be in the sample? How many will be missing (OOB)?
3. **Hyperparameter Behavior:** Explain why increasing `n_estimators` reduces variance but doesn't cause overfitting, while increasing `max_depth` can cause overfitting.
4. **OOB vs. Cross-Validation:** What advantage does OOB error have over 5-fold cross-validation?
5. **The Correlation Problem:** Why might all trees in a bagging ensemble start with the same root node? How does this affect the variance reduction?
6. **Practical Question:** You train a bagging classifier with 100 trees. The training accuracy is 99%, but the OOB score is 85%. What does this tell you? What might you try to improve?
7. **Code Question:** Write sklearn code to:
   - Create a BaggingRegressor with 200 trees
   - Use trees of depth 10
   - Enable OOB scoring
   - Print both OOB score and test score

■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 22: Random Forests

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Objective:** Understand how Random Forests decorrelate trees, learn variable importance methods, and handle class imbalance and missing data

Contents

1 The Problem with Bagging: Correlated Trees

Key Summary

Random Forest is an enhancement of Bagging that adds **feature randomization** at each split to reduce correlation between trees. This lecture covers:

1. Why bagged trees are often correlated and why that's a problem
2. How Random Forests decorrelate trees
3. Variable importance methods (MDI and Permutation)
4. Handling class imbalance
5. Missing data with surrogate splits

1.1 Quick Review: Bagging

Recall from last lecture:

- **Bagging** = Bootstrap + Aggregating
- Create B bootstrap samples from your training data
- Train a decision tree on each sample
- Aggregate predictions (majority vote or average)
- Use **OOB error** for validation

The key insight was that averaging multiple predictions reduces variance by a factor related to $1/\sqrt{n}$.

1.2 The Correlation Problem

Here's the issue we discovered: **Bagged trees often look very similar**.

Example: Why Trees Are Correlated

Imagine you have a dataset with 10 features, and one of them—`Glucose`—is an extremely strong predictor of diabetes.

When you build Tree 1 on Bootstrap Sample 1:

- You check all 10 features for the best split
- `Glucose` wins (it always does—it's the strongest!)
- Tree 1's root node splits on `Glucose`

When you build Tree 2 on Bootstrap Sample 2:

- You check all 10 features for the best split
- `Glucose` wins again!
- Tree 2's root node also splits on `Glucose`

This pattern continues for all B trees. They all start the same way!

Important: Why Correlation Matters

The variance reduction formula assumes predictions are **independent**:

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n} \quad \text{(if independent)}$$

But if predictions are **correlated**, the variance reduction is weaker:

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n} + \rho\sigma^2 \cdot \frac{n-1}{n} \approx \rho\sigma^2 \quad (\text{as } n \to \infty)$$

where ρ is the correlation. High correlation means less variance reduction—exactly what we don't want!

Example: The Doctor Analogy Revisited

Remember our analogy of getting multiple medical opinions? If all doctors were trained at the same hospital with the same textbooks, they'd give you the same advice: "Rest, drink water, come back in 3 days."

That's not very useful! You want doctors with **diverse training** and **different perspectives**. The same applies to trees—we want them to be diverse, not copies of each other.

2 The Random Forest Solution: Feature Randomization

Definition: Random Forest

A **Random Forest** is a modification of Bagging where, at **each split in each tree**, we only consider a **random subset of features** for splitting.
This forces trees to make different choices, reducing correlation.

2.1 How Random Forests Work

The algorithm is beautifully simple—just one change from bagging:

1. Create B bootstrap samples (same as bagging)
2. For each tree, at **each split**:
   (a) Randomly select m features from the total p features (where $m < p$)
   (b) Find the best split among **only these m features**
   (c) Make the split
3. Repeat until trees are fully grown (or reach stopping criteria)
4. Aggregate predictions (same as bagging)

Important: The Key Difference

Bagging: At each split, consider **all p features** → strong features always win
Random Forest: At each split, consider **only m features** → strong features sometimes excluded
This "controlled randomness" forces diversity!

Example: Random Forest in Action

Dataset: Heart disease prediction with 5 features:
Age, Sex, Max Heart Rate, Cholesterol, Chest Pain
Setting: $m = 3$ (consider 3 random features at each split)
Building Tree 1:

- *Root split*: Randomly select 3 features → {Age, Sex, Cholesterol}
- Best split among these 3: **Sex**
- *Next split (left child)*: Randomly select 3 features → {Age, Max HR, Chest Pain}
- Best split among these 3: **Max Heart Rate**
- Continue...

Building Tree 2:

- *Root split*: Randomly select 3 features → {Cholesterol, Chest Pain, Max HR}
- Best split among these 3: **Chest Pain**
- Different root! Trees will be less correlated.

Notice: Tree 1 and Tree 2 start with different splits because they considered different feature subsets!

2.2 Choosing m: The Number of Features to Consider

The parameter m (often called `max_features` in sklearn) controls how many features we randomly select at each split.

Definition: Rules of Thumb for m

For Classification:

$$m \approx \sqrt{p}$$

where p is the total number of features.

For Regression:

$$m \approx \frac{p}{3}$$

These are starting points—tune with cross-validation or OOB error!

Example: Choosing m in Practice

If you have 100 features:

- **Classification**: Start with $m = \sqrt{100} = 10$
- **Regression**: Start with $m = 100/3 \approx 33$

If you have 9 features (like the Diabetes dataset):

- **Classification**: Start with $m = \sqrt{9} = 3$
- **Regression**: Start with $m = 9/3 = 3$

2.3 Extreme Cases: What Happens with Different m?

Understanding the extremes helps build intuition:

- $m = p$ **(all features)**: This is just regular Bagging! Trees are highly correlated.
- $m = 1$ **(one feature)**: Completely random trees. Each split is determined by chance. Trees are uncorrelated but individually very weak.
- $m = \sqrt{p}$ **or** $m = p/3$: The sweet spot. Trees are diverse but still make reasonably good splits.

3 Random Forest Hyperparameters

Random Forests have more hyperparameters than single trees. Let's organize them:

3.1 Overview of Hyperparameters

Table 1: *Random Forest Hyperparameters*

Parameter	Controls	Notes
`n_estimators`	Number of trees (B)	More is better, never overfits
`max_features`	Features per split (m)	Key decorrelation parameter
`max_depth`	Tree depth	Controls tree complexity
`min_samples_split`	Min samples to split	Stopping condition
`min_samples_leaf`	Min samples per leaf	Stopping condition
`criterion`	Gini or Entropy	Usually doesn't matter much

3.2 Practical Advice for Tuning

Key Information

Expert Tips from the Instructors:

1. **Start with defaults**: sklearn's defaults are usually reasonable
2. **Focus on the important ones**:
   - `n_estimators`: Start with 100-500. More is always safe (no overfitting)
   - `max_features`: Start with $\sqrt{p}$ for classification, $p/3$ for regression
   - `max_depth`: Try None (fully grown) first, then limit if needed
3. **Don't overthink criterion**: Gini and Entropy give similar results. Just pick Gini.
4. **Use OOB instead of CV**: Random Forests give you OOB error for free—use it!

3.3 The Hyperparameter Explosion Problem

With multiple hyperparameters, the search space explodes:

- 5 choices for `max_features` $\times$
- 5 choices for `max_depth` $\times$
- 5 choices for `n_estimators` $\times$
- 2 choices for `criterion`
- $= 250$ combinations!

With 5-fold CV, that's 1,250 models to train. This is why:

1. We use rules of thumb to narrow the search
2. We use OOB error instead of CV (no additional training needed)
3. We use random search instead of grid search

4 Variable Importance

When we moved from single decision trees to ensembles, we lost **interpretability**. You can no longer say "if Glucose > 120 and Age > 50, then Diabetic" because you have hundreds of different trees!

Variable Importance is how we get some interpretability back.

4.1 Why Variable Importance Matters

Example: Motivating Example

You build a Random Forest to predict customer churn with 50 features. The model achieves 95% accuracy. Your boss asks:
"Great accuracy, but **why** are customers leaving? What should we focus on?"
Variable importance answers this: "The top 3 factors are: Contract Type, Monthly Charges, and Tenure. Customers with month-to-month contracts and high monthly charges who've been with us less than a year are most likely to leave."
Now you have actionable insights!

4.2 Method 1: Mean Decrease in Impurity (MDI)

Definition: Mean Decrease in Impurity (MDI)

MDI measures how much each feature contributes to reducing impurity (Gini or MSE) across all trees in the forest.
For each feature:
1. Find all nodes in all trees where that feature is used for splitting
2. Sum up the impurity decrease at each of those nodes
3. Average across all trees

Features that cause larger decreases in impurity are more important.

4.2.1 MDI Calculation Step by Step

1. **For each node n that uses feature j:**

$$\Delta I_n = \frac{n_{samples}}{N} \times \left[I_{before} - \sum_{c \in children} \frac{n_c}{n_{samples}} I_c \right]$$

   where I is impurity (Gini), $n_{samples}$ is samples at node, N is total samples

2. **Sum for feature j in tree t:**

$$F_j^{(t)} = \sum_{\text{nodes using } j} \Delta I_n$$

3. **Normalize within each tree:**

$$\hat{F}_j^{(t)} = \frac{F_j^{(t)}}{\sum_k F_k^{(t)}}$$

4. **Average across all trees:**

$$\text{MDI}_j = \frac{1}{B}\sum_{t=1}^{B} \hat{F}_j^{(t)}$$

4.2.2 MDI Pros and Cons

+ **Fast**: Computed during training, no extra computation
+ **Built into sklearn**: Just access `model.feature_importances_`
– **Biased toward high-cardinality features**: Features with many unique values (continuous or categorical with many categories) get artificially inflated importance
– **Computed on training data**: Can be misleading if model overfits

4.3 Method 2: Permutation Importance

Definition: Permutation Importance

Permutation importance measures how much model performance **decreases** when a feature's values are randomly shuffled.
If shuffling a feature destroys model accuracy, that feature was important!

4.3.1 Permutation Importance Algorithm

1. **Baseline**: Compute baseline OOB accuracy $s_{baseline}$
2. **For each feature** j:
   (a) Randomly shuffle (permute) the values of feature j
   (b) Compute OOB accuracy with shuffled feature: $s_j^{(k)}$
   (c) Repeat K times and average: $s_j = \frac{1}{K}\sum_k s_j^{(k)}$
3. **Calculate importance**:

$$\text{Importance}_j = s_{baseline} - s_j$$

Example: Permutation Importance Intuition

Dataset: Predicting fitness level from height, weight, and age
Baseline OOB accuracy: 88%
Shuffle height:
- Original: $[170, 165, 180, 175, 160]$
- Shuffled: $[175, 180, 165, 160, 170]$
- New accuracy: 87%
- Importance = 88% - 87% = 1% (height matters a little)

Shuffle weight:
- New accuracy: 72%
- Importance = 88% - 72% = 16% (weight matters A LOT!)

Conclusion: Weight is much more important than height for predicting fitness.

4.3.2 Permutation Importance Pros and Cons

+ **Not biased**: Works equally well for all feature types
+ **Intuitive**: Directly measures impact on model performance
+ **Uses validation data**: Reflects true generalization, not training fit
– **Slower**: Requires multiple predictions per feature
– **Correlated features**: Can underestimate importance when features are correlated

4.4 MDI vs. Permutation: Which to Use?

Important: Recommendation

Use Permutation Importance when possible.
MDI is convenient (it's built into sklearn and fast), but it's biased toward high-cardinality features. Permutation importance gives more reliable results.
In practice:

- Use MDI for quick exploration
- Use Permutation Importance for final analysis and reporting

4.5 Comparing Bagging vs. Random Forest Importance

An interesting observation: variable importance looks different in Bagging vs. Random Forest!

- **Bagging**: A few features dominate (because trees are correlated)
- **Random Forest**: Importance is spread across more features (trees are diverse)

This smoother distribution in Random Forest reflects the fact that trees are considering different features, leading to a more complete picture of which features matter.

5 When Random Forest Doesn't Work Well

Random Forest is powerful, but not perfect. Here's when it struggles:

Warning

Too Many Irrelevant Features

If you have 1000 features but only 10 are actually useful, Random Forest may perform poorly.

Why? At each split, we randomly select m features. If most features are junk, there's a high probability that our random subset contains mostly junk features.

Solution: Use feature selection or PCA to reduce dimensionality first. Then apply Random Forest.

Example: When Bagging Beats Random Forest

You might encounter this surprising result:

Method	OOB Accuracy
Random Forest	78%
Bagging	82%

This often happens when:

- You have many noisy/irrelevant features
- Only a few features are truly predictive
- Random Forest keeps "missing" the good features due to random selection

In this case, remove irrelevant features and try again!

6 Handling Class Imbalance

Class imbalance occurs when one class has far more samples than another. This is extremely common in real-world problems.

Example: Class Imbalance Examples

- **Fraud detection**: 99.9% legitimate transactions, 0.1% fraudulent
- **Disease diagnosis**: 99% healthy, 1% have the disease
- **Spam detection**: 80% legitimate emails, 20% spam
- **Manufacturing defects**: 99.5% good products, 0.5% defective

6.1 Why Accuracy is Misleading

Warning

Accuracy can be useless for imbalanced data!
If 99% of emails are not spam, a model that predicts "not spam" for everything achieves 99% accuracy—but catches zero spam.
Solution: Use better metrics like F1-score, AUC-ROC, or precision/recall.

6.2 Three Approaches to Handle Imbalance

6.2.1 1. Undersampling (Reduce Majority Class)

Definition: Undersampling

Remove samples from the majority class until classes are balanced.

Methods:

- **Random Undersampling**: Randomly remove majority class samples
- **Near Miss**: Remove majority samples that are **far** from the decision boundary (keep the "hard" examples near the boundary)

Pros: Reduces training time
Cons: Throws away potentially useful data

6.2.2 2. Oversampling (Increase Minority Class)

Definition: Oversampling

Create more samples of the minority class until classes are balanced.

Methods:

- **Random Oversampling**: Duplicate minority samples (bootstrap with replacement)
- **SMOTE** (Synthetic Minority Oversampling Technique): Create **synthetic** samples by interpolating

between existing minority samples

Example: SMOTE Intuition

Instead of just copying minority samples:

1. Pick a minority sample x_i
2. Find its k nearest neighbors (also minority class)
3. Randomly pick one neighbor x_j
4. Create a new sample along the line between x_i and x_j:

$$x_{new} = x_i + \lambda \cdot (x_j - x_i) \quad \text{where } \lambda \in [0, 1]$$

This creates new, plausible minority samples that expand the feature space rather than just duplicating existing points.

6.2.3 3. Class Weighting

Definition: Class Weighting

Modify the loss function to penalize errors on the minority class more heavily.

Instead of treating all misclassifications equally, we weight them:

$$w_k = \frac{N}{K \times N_k}$$

where:

- N = total samples
- K = number of classes
- N_k = samples in class k

In sklearn, just set `class_weight='balanced'`:

```
from sklearn.ensemble import RandomForestClassifier

# Automatically adjusts weights based on class frequencies
rf = RandomForestClassifier(
    n_estimators=100,
    class_weight='balanced',  # This handles imbalance!
    random_state=42
)
```

6.3 Practical Advice for Imbalanced Data

Key Information

When to use which approach:

- **Mild imbalance (60/40)**: Class weighting is often enough
- **Moderate imbalance (90/10)**: Try SMOTE or combination of over/undersampling
- **Severe imbalance (99/1)**: May need domain-specific techniques or anomaly detection

General rule: Always address imbalance! Even 40/60 splits benefit from handling.

7 Missing Data: Surrogate Splits

Decision trees have a unique way of handling missing data called **surrogate splits**.

7.1 The Problem

During prediction, what happens when a test sample has a missing value for the feature used to split a node?

Example: The Missing Data Problem

Your trained tree splits on `Blood Pressure` at the root. A new patient arrives, but their blood pressure wasn't recorded.
How do you route this patient through the tree?

7.2 The Solution: Surrogate Splits

Definition: Surrogate Split

A **surrogate split** is a backup split that produces similar results to the primary split.
During training, for each split on feature X, we find other features that would split the data similarly. These become surrogates.

7.2.1 How Surrogate Splits Work

1. **During Training**:
   (a) Find the best split (e.g., `Arteries Blocked > 2`)
   (b) Record how samples are distributed after this split
   (c) For each other feature, find the split that most closely mimics this distribution
   (d) Rank these alternative splits by similarity
2. **During Prediction**:
   (a) If the primary feature has a missing value, use the best surrogate
   (b) If that's also missing, try the second surrogate
   (c) Continue down the list

Example: Surrogate Split Example

Primary split: `Arteries Blocked`

- FALSE → 3 No, 1 Yes (heart disease)
- TRUE → 0 No, 2 Yes

Finding surrogates—test other features:

`Chest Congestion`:

- FALSE → 3 No, 2 Yes
- TRUE → 0 No, 1 Yes

Similarity: Very similar distribution! Only 2 "flips" needed.

`Good Blood Circulation`:

- FALSE → 2 No, 3 Yes
- TRUE → 1 No, 0 Yes

Similarity: Less similar, 6 "flips" needed.

Result: `Chest Congestion` becomes the primary surrogate for `Arteries Blocked`.

If a patient's `Arteries Blocked` value is missing, we use their `Chest Congestion` value instead!

7.3 Benefits of Surrogate Splits

- **No imputation needed**: Tree handles missing data automatically
- **Interpretable**: Surrogates show which features are related
- **Works well with correlated features**: Multicollinearity actually helps find good surrogates!

Warning

Surrogate splits work best when features are correlated. If features are independent, good surrogates may not exist, and you'll need other imputation methods.

8 Random Forests in Python

8.1 Basic Implementation

```python
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance
import numpy as np

# Load your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create Random Forest classifier
rf_clf = RandomForestClassifier(
    n_estimators=100,         # Number of trees
    max_features='sqrt',      # Features per split (sqrt for classification)
    max_depth=None,           # Grow trees fully
    min_samples_leaf=1,       # Minimum samples per leaf
    oob_score=True,           # Compute OOB score
    class_weight='balanced', # Handle class imbalance
    n_jobs=-1,                # Use all CPU cores
    random_state=42
)

# Train
rf_clf.fit(X_train, y_train)

# Results
print(f"OOB Score: {rf_clf.oob_score_:.4f}")
print(f"Test Accuracy: {rf_clf.score(X_test, y_test):.4f}")
```

8.2 Getting Variable Importance

```python
# Method 1: MDI (built-in, but biased)
mdi_importance = rf_clf.feature_importances_

# Method 2: Permutation Importance (preferred)
perm_importance = permutation_importance(
    rf_clf, X_test, y_test,
    n_repeats=10,
    random_state=42
)

# Display results
import pandas as pd

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'MDI': mdi_importance,
```

```
    'Permutation': perm_importance.importances_mean
}).sort_values('Permutation', ascending=False)

print(importance_df)
```

8.3 Handling Class Imbalance with SMOTE

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

# Create sampling pipeline
over = SMOTE(sampling_strategy=0.5)  # Oversample minority to 50%
under = RandomUnderSampler(sampling_strategy=0.8)  # Then undersample

# Apply to training data
X_resampled, y_resampled = over.fit_resample(X_train, y_train)
X_resampled, y_resampled = under.fit_resample(X_resampled, y_resampled)

# Train on balanced data
rf_clf.fit(X_resampled, y_resampled)
```

9 Key Takeaways

Key Summary

Random Forest:

- Enhancement of Bagging that adds feature randomization at each split
- At each split, randomly select m features and find best split among them
- Rule of thumb: $m = \sqrt{p}$ for classification, $m = p/3$ for regression
- Reduces tree correlation → better variance reduction

Key Hyperparameters:

- `n_estimators`: More is always better (no overfitting risk)
- `max_features`: Controls decorrelation
- Use OOB error for validation

Variable Importance:

- MDI: Fast but biased toward high-cardinality features
- Permutation: Slower but more reliable
- Use permutation importance for final analysis

Class Imbalance:

- Don't use accuracy! Use F1-score or AUC
- Options: Undersampling, Oversampling (SMOTE), Class weighting
- Always address imbalance, even for 40/60 splits

Missing Data:

- Surrogate splits provide automatic handling
- Find backup features that split similarly to primary feature
- Works best with correlated features

Table 2: *Comparison: Bagging vs. Random Forest*

Aspect	Bagging	Random Forest
Features per split	All p	Random subset m
Tree correlation	High	Low
Variance reduction	Good	Better
Importance distribution	Concentrated	Spread out
Works with many junk features	Better	May struggle

10 Practice Questions

1. **Decorrelation**: Explain why selecting a random subset of features at each split reduces correlation between trees.
2. **Hyperparameter Impact**: What happens to Random Forest if you set `max_features` equal to the total number of features? How does this compare to Bagging?
3. **MDI Bias**: You have two features: "Age" (continuous, 80 unique values) and "Gender" (binary, 2 values). Using MDI, which feature is likely to appear more important even if they have equal true importance? Why?
4. **Class Imbalance**: You're building a fraud detection model where only 0.5% of transactions are fraudulent. What metric should you use instead of accuracy? What technique(s) would you apply to handle the imbalance?
5. **Surrogate Splits**: Explain how surrogate splits help when a test sample has a missing value. When might surrogate splits fail to work well?
6. **Practical Scenario**: You build a Random Forest with 100 features, but only 5 are truly predictive. The OOB accuracy is 70%. You try Bagging and get 82%. Why might Bagging outperform Random Forest here?
7. **Code Question**: Write sklearn code to:
   - Train a Random Forest Regressor with OOB scoring enabled
   - Use $m = p/3$ features per split
   - Calculate and display permutation importance

■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 23: Gradient Boosting

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Objective:** Understand boosting as an approach to reduce bias, learn how gradient boosting works, and connect it to gradient descent

Contents

1 The Big Picture: A Different Approach to Ensembles

Key Summary

This lecture introduces **Boosting**, a fundamentally different ensemble approach from Bagging and Random Forests:

- **Random Forest**: Start with complex trees (low bias, high variance) → reduce **variance** by averaging
- **Boosting**: Start with simple trees (high bias, low variance) → reduce **bias** by iterative correction

The key insight: **Learn from your mistakes**. Each new tree corrects the errors of all previous trees.

1.1 Where We Are in the Course

Let's recap our journey through tree-based methods:

1. **Single Decision Trees**: Simple, interpretable, but prone to overfitting or underfitting
2. **Bagging**: Reduce variance by averaging many deep trees trained on bootstrap samples
3. **Random Forest**: Improve bagging by decorrelating trees (random feature selection)
4. **Boosting**: Take a completely different approach—reduce bias instead of variance

1.2 Why Boosting Matters

Key Information

The Practical Importance of Boosting:

For **tabular data** (spreadsheets, databases—not images or text), tree-based ensemble methods often outperform deep learning:

- On Kaggle competitions with structured data, boosting methods (especially XGBoost, LightGBM, CatBoost) win **almost every time**
- Random Forests and Boosting are the "go-to" methods for tabular problems
- Neural networks excel at images, text, and audio—but for tables, trees reign supreme

This is why understanding boosting is essential for any data scientist!

2 The Intuition Behind Boosting

2.1 The Opposite of Random Forest

Random Forest and Boosting take opposite approaches to the bias-variance tradeoff:

Table 1: *Random Forest vs. Boosting*

Aspect	Random Forest	Boosting
Starting point	Deep trees (complex)	Shallow trees (simple)
Initial problem	High variance	High bias
Solution	Average many trees	Add trees sequentially
Goal	Reduce variance	Reduce bias
Training	Parallel (independent)	Sequential (dependent)

2.2 The Exam Analogy

Example: Passing a Hard Exam

Imagine you need to pass a very difficult exam (say, you need an A to pass). You could:

Option 1 (Bad): Steal a time machine, go back to 1996, befriend the inventors of boosting, study with them for a decade, return to present. (Not practical!)

Option 2 (Good): Gather simple rules of thumb from different people:

1. **Last year's student**: "Never choose option D" → You get 60%
2. **Teaching Assistant**: "If 'overfitting' is an option, it's usually correct" → Focus on questions you got wrong, now at 65%
3. **Professor**: "Cross-validation is always a good answer" → Focus on remaining mistakes, now at 70%

Combine the rules with appropriate weights:

- Give more weight to more reliable rules
- Each rule is a "weak learner" (not great on its own)
- Combined, they form a "strong learner" (90%+ accuracy!)

This is boosting! Simple rules, learned sequentially, each fixing the mistakes of previous ones.

2.3 Key Concepts

Definition: Weak Learner

A **weak learner** is a model that performs only slightly better than random guessing. In the context of trees, this typically means a **stump**—a tree with only one split (depth 1).

The remarkable insight: combining many weak learners can create a **strong learner**!

Definition: Boosting

Boosting is an ensemble method that:

1. Uses **simple models** (weak learners) as building blocks
2. Combines them **additively**: $T_{final}(x) = \sum_h \lambda_h T_h(x)$
3. Builds models **sequentially**, with each new model correcting previous mistakes

The three keywords to remember:

1. **Simple**: Use weak learners (stumps)
2. **Additive**: Add models together
3. **Sequential**: Each model learns from previous mistakes

3 How Gradient Boosting Works

Key Summary

The core idea of gradient boosting is beautifully simple:

Each new tree predicts the residuals (errors) of the current ensemble.

If tree 1 predicts "too low by 5," tree 2 learns to predict "+5" to correct it.

3.1 The Algorithm Step by Step

Definition: Gradient Boosting Algorithm (Regression)

Input: Training data (x_i, y_i), learning rate λ, number of iterations M

Initialize: $T_0(x) =$ simple model (e.g., predicting the mean of y)

For $m = 1, 2, \ldots, M$**:**

1. **Compute residuals**: $r_i^{(m-1)} = y_i - T_{m-1}(x_i)$
   (These are the "mistakes" of the current model)
2. **Fit new tree to residuals**: Train a simple tree t_m to predict $r^{(m-1)}$
3. **Update model**: $T_m(x) = T_{m-1}(x) + \lambda \cdot t_m(x)$

Output: $T_M(x) = T_0(x) + \lambda t_1(x) + \lambda t_2(x) + \cdots + \lambda t_M(x)$

3.2 A Concrete Example

Example: Gradient Boosting on Simple Data

Data: 6 points with input x and target y

Step 1: Fit initial model T_0

- Use a stump (one split)
- Find the split that minimizes MSE
- Say we split at $x = 6.5$:
  - Left region ($x < 6.5$): Predict mean ≈ 0
  - Right region ($x \geq 6.5$): Predict mean ≈ 7.5

Step 2: Compute residuals

- $r_i = y_i - T_0(x_i)$
- These show where our model is wrong

Step 3: Fit t_1 to residuals

- Train a stump to predict the residuals
- Say we split at $x = 3.5$:
  - Left region: Predict mean residual ≈ 2
  - Right region: Predict mean residual ≈ -1.5

Step 4: Update model

- Set $\lambda = 0.5$ (learning rate)

- $T_1(x) = T_0(x) + 0.5 \cdot t_1(x)$

Step 5: Repeat

- Compute new residuals from T_1
- Fit t_2 to these residuals
- $T_2(x) = T_1(x) + 0.5 \cdot t_2(x)$
- Continue until stopping criterion

3.3 Why the Learning Rate (λ)?

A natural question: Why not just add the full residual prediction? Why multiply by $\lambda < 1$?

Important: The Learning Rate

The learning rate λ controls **how much we trust each new tree**.

Problem without λ:

- We're fitting residuals with a **weak learner** (stump)
- The stump's prediction of residuals is **imperfect**
- If we add the full prediction, we might **overcorrect**
- This is like playing whack-a-mole: fix one error, create another

Solution with λ:

- Take a **small step** in the right direction
- Don't fully trust any single weak learner
- Gradually converge to the correct answer
- Typical values: $\lambda = 0.01$ to 0.1

Think of it like this: if you're not sure of the direction, take small steps rather than giant leaps!

3.4 Important Implementation Note

Warning

We don't actually combine trees into one big tree!

In practice:

- We store each tree separately: $T_0, t_1, t_2, \ldots, t_M$
- For prediction, we evaluate each tree and sum the results:

$$\hat{y} = T_0(x) + \lambda \cdot t_1(x) + \lambda \cdot t_2(x) + \cdots$$

- This is just numerical addition—no tree merging!

4 Why Is It Called "Gradient" Boosting?

This is the mathematical heart of the lecture. Understanding this connection reveals why boosting works so well.

4.1 Review: Gradient Descent

Definition: Gradient Descent

Gradient descent is an iterative optimization algorithm:
Goal: Find parameters w that minimize a loss function $L(w)$
Update rule:

$$w_{new} = w_{old} - \lambda \cdot \nabla L(w)$$

where:

- $\nabla L(w)$ is the gradient (direction of steepest increase)
- λ is the learning rate (step size)
- We move in the **opposite** direction of the gradient (toward minimum)

Example: Gradient Descent Intuition

Imagine you're blindfolded on a mountain and want to reach the valley:

1. **Feel the slope**: Figure out which direction is "up" (the gradient)
2. **Step downhill**: Move in the opposite direction
3. **Choose step size**: If the slope is steep, take a bigger step; if gentle, take a smaller step
4. **Repeat**: Keep feeling the slope and stepping until you reach the bottom

The learning rate controls your step size:

- Too large: You might overstep and end up on the other side
- Too small: You'll get there eventually, but very slowly

4.2 The Key Connection: Residuals ARE Gradients

Here's the magical connection that justifies the name "gradient boosting."

Consider the MSE loss function:

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

Take the derivative with respect to the prediction $\hat{y}$:

$$\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y})$$

Important: The Key Insight

The **negative gradient** of MSE with respect to predictions is exactly the **residual**:

$$-\frac{\partial L}{\partial \hat{y}} = y - \hat{y} = \text{residual}$$

Therefore:

- When we fit a tree to **residuals**, we're fitting it to the **negative gradient**
- Adding $\lambda \times$ (residual prediction) is exactly gradient descent!
- We're doing gradient descent in **function space** rather than parameter space

4.3 Why This Matters

Understanding the gradient descent connection gives us:

1. **Theoretical guarantee**: If the loss is convex and learning rate is small enough, gradient descent converges to the minimum. So our boosting algorithm will converge!
2. **Extension to other losses**: We can use ANY differentiable loss function:
   - MSE for regression
   - Log-loss for classification (this gives AdaBoost-like behavior)
   - Huber loss for robust regression
   - Quantile loss for quantile regression

   Just compute the negative gradient and fit trees to that!
3. **Understanding the learning rate**: The learning rate in boosting plays the same role as in gradient descent—controlling step size to ensure convergence.

4.4 Optimization in Function Space

Example: Parameter Space vs. Function Space

Traditional ML (Linear/Logistic Regression):

- We have parameters $\beta_0, \beta_1, \ldots$
- We optimize by adjusting these parameters
- Gradient descent moves in **parameter space**

Gradient Boosting:

- Trees don't have fixed parameters (depth can vary)
- We optimize the **predictions** directly
- Each tree moves us toward better predictions
- Gradient descent moves in **function/prediction space**

This is a profound shift! Instead of adjusting coefficients, we're directly adjusting predictions by adding corrective models.

5 Hyperparameters and Tuning

Gradient boosting has several important hyperparameters to tune.

5.1 Learning Rate (λ)

Table 2: *Effect of Learning Rate*

Learning Rate	Pros	Cons
Small (0.01)	More robust, better generalization	Needs many trees, slow
Medium (0.1)	Good balance	Standard choice
Large (0.3+)	Faster training	May overshoot, overfit

Rule of thumb: Use a small learning rate with many trees for best performance.

5.2 Number of Trees (Iterations)

Unlike Random Forest, boosting CAN overfit with too many trees:

- **Too few**: Underfitting (high bias)
- **Too many**: Overfitting (memorizing training data)
- **Solution**: Use early stopping with validation data

5.3 Tree Complexity

Each weak learner can have its own complexity:

- **Depth 1 (stump)**: Very weak, needs many trees
- **Depth 3-5**: Common choice, captures interactions
- **Depth 10+**: May overfit, defeats purpose of "weak" learners

5.4 Practical Tuning Strategy

Key Information

Recommended Tuning Approach:

1. Start with:
   - Learning rate: 0.1
   - Max depth: 3
   - N estimators: 100
2. Watch the learning curve (training vs. validation error)
3. If overfitting: Decrease learning rate, add regularization
4. If underfitting: Increase depth or number of trees
5. Final model: Use small learning rate (0.01-0.05) with early stopping

6 Gradient Boosting in Python

6.1 Basic Implementation with sklearn

```python
from sklearn.ensemble import GradientBoostingRegressor, GradientBoostingClassifier
from sklearn.model_selection import train_test_split

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create gradient boosting regressor
gb_reg = GradientBoostingRegressor(
    n_estimators=100,       # Number of trees
    learning_rate=0.1,      # Shrinkage parameter
    max_depth=3,            # Depth of each tree (weak learner)
    min_samples_split=2,    # Minimum samples to split
    min_samples_leaf=1,     # Minimum samples per leaf
    random_state=42
)

# Train
gb_reg.fit(X_train, y_train)

# Evaluate
train_score = gb_reg.score(X_train, y_train)
test_score = gb_reg.score(X_test, y_test)
print(f"Train R^2: {train_score:.4f}")
print(f"Test R^2: {test_score:.4f}")
```

6.2 Monitoring Training with Staged Predict

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# Get staged predictions
train_errors = []
test_errors = []

for i, y_pred_train in enumerate(gb_reg.staged_predict(X_train)):
    train_errors.append(mean_squared_error(y_train, y_pred_train))

for i, y_pred_test in enumerate(gb_reg.staged_predict(X_test)):
    test_errors.append(mean_squared_error(y_test, y_pred_test))

# Plot learning curve
plt.figure(figsize=(10, 6))
plt.plot(train_errors, label='Training Error')
```

```
plt.plot(test_errors, label='Test Error')
plt.xlabel('Number of Trees')
plt.ylabel('MSE')
plt.title('Gradient Boosting Learning Curve')
plt.legend()
plt.show()

# Find optimal number of trees
best_n = np.argmin(test_errors)
print(f"Optimal number of trees: {best_n}")
```

6.3 Early Stopping

```
from sklearn.ensemble import GradientBoostingRegressor

# Use validation_fraction and n_iter_no_change for early stopping
gb_early = GradientBoostingRegressor(
    n_estimators=500,           # Maximum trees (won't necessarily use all)
    learning_rate=0.1,
    max_depth=3,
    validation_fraction=0.1,    # Use 10% for validation
    n_iter_no_change=10,        # Stop if no improvement for 10 rounds
    tol=1e-4,                   # Tolerance for improvement
    random_state=42
)

gb_early.fit(X_train, y_train)
print(f"Trees used: {gb_early.n_estimators_}")
```

6.4 Feature Importance

```
import pandas as pd

# Get feature importances (based on improvement in criterion)
importances = gb_reg.feature_importances_

# Display sorted importances
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values('Importance', ascending=False)

print(feature_importance_df)

# Plot
plt.figure(figsize=(10, 6))
plt.barh(range(len(importances)), importances[importances.argsort()])
plt.yticks(range(len(importances)),
```

```
            [feature_names[i] for i in importances.argsort()])
plt.xlabel('Feature Importance')
plt.title('Gradient Boosting Feature Importance')
plt.show()
```

7 Comparing Ensemble Methods

Table 3: *Comprehensive Comparison of Tree Ensemble Methods*

Aspect	**Bagging**	**Random Forest**	**Gradient Boosting**
Primary goal	Reduce variance	Reduce variance	Reduce bias
Tree type	Deep, complex	Deep, complex	Shallow, simple
Training	Parallel	Parallel	Sequential
Speed	Fast	Fast	Slower
Overfitting risk	Low	Low	Higher
Tuning difficulty	Easy	Easy	Harder
Feature randomization	No	Yes (at each split)	No
Tree correlation	High	Low	N/A (sequential)
Interpretability	Low	Low (some via importance)	Low (some via importance)
Performance	Good	Better	Often best

Key Information

When to Use Which:

- **Random Forest**: When you want good results with minimal tuning. Great default choice.
- **Gradient Boosting**: When you need maximum performance and are willing to tune carefully. Especially for competitions.
- **Both**: Always try both! The best method depends on your specific data.

8 Key Takeaways

Key Summary

Boosting Philosophy:

- Combine many **weak learners** into one **strong learner**
- Reduce **bias** (opposite of Random Forest which reduces variance)
- **Learn from mistakes**: Each new model corrects previous errors

Gradient Boosting Mechanics:

- Start with a simple prediction
- Compute residuals (errors)
- Fit new tree to predict residuals
- Add new tree with learning rate: $T_{new} = T_{old} + \lambda \cdot t_{new}$
- Repeat

The Gradient Connection:

- Residuals ARE the negative gradient of MSE loss
- Fitting trees to residuals = gradient descent in function space
- Learning rate = step size in gradient descent
- This is why it's called "gradient" boosting!

Hyperparameters:

- Learning rate (λ): Small values (0.01-0.1) are safer
- Number of trees: Use early stopping to prevent overfitting
- Tree depth: Keep trees shallow (depth 3-5)

Practical Advice:

- Gradient Boosting often outperforms Random Forest (with tuning)
- For tabular data, tree methods often beat deep learning
- XGBoost, LightGBM, CatBoost are optimized implementations

9 Practice Questions

1. **Conceptual**: Explain the fundamental difference between how Random Forest and Gradient Boosting reduce prediction error. Which component of error does each target?
2. **Algorithm**: In gradient boosting, what is the "target" that each new tree is trained to predict? Why is this different from the original target y?
3. **Mathematical**: Show that for MSE loss $L = \frac{1}{2}(y - \hat{y})^2$, the negative gradient with respect to $\hat{y}$ equals the residual.
4. **Learning Rate**: Why do we use a learning rate $\lambda < 1$ instead of adding the full residual prediction? What would happen if λ were too large?
5. **Overfitting**: Unlike Random Forest, Gradient Boosting can overfit with too many trees. Why is this the case? How does early stopping help?
6. **Comparison**: You have a dataset and find that:
   - Random Forest: Train accuracy 95%, Test accuracy 88%
   - Gradient Boosting: Train accuracy 99%, Test accuracy 85%

   What does this suggest about your Gradient Boosting model? What would you adjust?
7. **Code**: Write Python code using sklearn to:
   - Train a GradientBoostingClassifier
   - Use early stopping based on validation performance
   - Plot the learning curve showing train vs. validation error
8. **Extension**: Gradient boosting works with any differentiable loss function. How would the algorithm change if we used absolute error loss $L = |y - \hat{y}|$ instead of MSE? What would we fit trees to?

■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 24: AdaBoost

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Objective:** Understand AdaBoost as a boosting algorithm for classification, learn how it differs from Gradient Boosting, and master the weight update mechanism

Contents

1 Introduction to AdaBoost

Key Summary

AdaBoost (Adaptive Boosting) is a boosting algorithm designed for classification. Its key innovation:

Instead of fitting residuals, AdaBoost reweights the training data.

After each weak learner:

- Misclassified samples get **higher weights**
- Correctly classified samples get **lower weights**
- The next weak learner focuses on the "hard" examples

This is like studying for an exam by focusing on the problems you got wrong!

1.1 Historical Context

AdaBoost was introduced by Yoav Freund and Robert Schapire in 1996 and won the prestigious Gödel Prize in 2003. It was one of the first practical boosting algorithms and demonstrated that weak learners could be combined into a strong learner.

1.2 The "Error Notebook" Analogy

Example: Studying with an Error Notebook

Imagine you're preparing for a difficult exam:

1. **Practice test 1**: You get some questions right, some wrong
2. **Star the wrong ones**: Mark the problems you missed with a star ($\star$)
3. **Focus your study**: Spend more time on starred problems
4. **Practice test 2**: You might get the starred ones right, but miss some new ones
5. **Update stars**: Star the new mistakes, remove stars from ones you now understand
6. **Repeat**: Keep focusing on your current weaknesses
7. **Final exam**: Combine everything you learned from all practice tests

This is exactly how AdaBoost works! Each weak learner is like a practice test, and the sample weights are like your star markings.

2 Key Concepts

2.1 Weak Learners: Stumps

Definition: Decision Stump

A **decision stump** is a decision tree with only **one split**:

- One root node (asks one question)
- Two leaf nodes (makes two predictions)

A stump is the simplest possible decision tree. It's a "weak learner"—better than random guessing, but not by much.

Why use such a simple model?

- Each stump captures **one simple pattern**
- Many stumps combined can capture **complex patterns**
- Simple models are **less prone to overfitting** individually
- The boosting process handles complexity through **aggregation**

2.2 Label Encoding: -1 and $+1$

AdaBoost uses labels $y \in \{-1, +1\}$ instead of $\{0, 1\}$. This makes the math elegant:

Definition: The Sign Trick

When $y, \hat{y} \in \{-1, +1\}$:

- If **correct**: $y \cdot \hat{y} = +1$ (both same sign)
- If **wrong**: $y \cdot \hat{y} = -1$ (opposite signs)

Examples:

- $y = +1$, $\hat{y} = +1 \Rightarrow y \cdot \hat{y} = +1$ (correct)
- $y = -1$, $\hat{y} = -1 \Rightarrow y \cdot \hat{y} = +1$ (correct)
- $y = +1$, $\hat{y} = -1 \Rightarrow y \cdot \hat{y} = -1$ (wrong)
- $y = -1$, $\hat{y} = +1 \Rightarrow y \cdot \hat{y} = -1$ (wrong)

This property is crucial for the weight update formula!

2.3 Sample Weights

Unlike gradient boosting (which modifies the target), AdaBoost maintains a **weight for each sample**:

- Initially, all samples have equal weight: $w_i = \frac{1}{N}$
- After each iteration, weights are adjusted:
  - Misclassified samples: weight **increases**
  - Correctly classified samples: weight **decreases**
- Weights always sum to 1 (they form a distribution)

3 The AdaBoost Algorithm

Definition: AdaBoost Algorithm

Input: Training data (x_i, y_i) with $y_i \in \{-1, +1\}$, number of iterations M

Initialize: $w_i^{(0)} = \frac{1}{N}$ for all samples

For $m = 1, 2, \ldots, M$**:**

1. **Fit weak learner** h_m using weights $w^{(m-1)}$
2. **Compute weighted error**:
$$\epsilon_m = \sum_{i: h_m(x_i) \neq y_i} w_i^{(m-1)}$$
(Sum of weights of misclassified samples)
3. **Compute learner weight** (how much "say" this learner gets):
$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$
4. **Update sample weights**:
$$w_i^{(m)} = w_i^{(m-1)} \cdot \exp(-\alpha_m \cdot y_i \cdot h_m(x_i))$$
Then normalize so weights sum to 1.
5. **Add to ensemble**:
$$H_m(x) = H_{m-1}(x) + \alpha_m \cdot h_m(x)$$

Final prediction: $\hat{y} = \text{sign}(H_M(x))$

3.1 Understanding the Learner Weight α

The formula $\alpha_m = \frac{1}{2} \ln \left(\frac{1-\epsilon_m}{\epsilon_m} \right)$ determines how much influence each weak learner has:

Table 1: *Learner Weight vs. Error Rate*

Error ϵ	**Meaning**	**Weight** α
0.0	Perfect classifier	$+\infty$
0.1	Very good	Large positive
0.3	Good	Moderate positive
0.5	Random guessing	0 (ignored!)
0.7	Worse than random	Negative (flips!)
1.0	Perfectly wrong	$-\infty$

Key Information

Key Insights:

- Better classifiers ($\epsilon < 0.5$) get positive weight—their votes count!
- Random classifiers ($\epsilon = 0.5$) get zero weight—ignored

- Worse-than-random classifiers ($\epsilon > 0.5$) get negative weight—their predictions are **flipped**!

3.2 Understanding the Weight Update

The weight update formula is:

$$w_i^{(new)} \propto w_i^{(old)} \cdot \exp(-\alpha \cdot y_i \cdot h(x_i))$$

Let's understand this:

- **If correctly classified**: $y_i \cdot h(x_i) = +1$
  - Exponent: $-\alpha \cdot (+1) = -\alpha < 0$
  - Factor: $e^{-\alpha} < 1$
  - Weight **decreases**
- **If misclassified**: $y_i \cdot h(x_i) = -1$
  - Exponent: $-\alpha \cdot (-1) = +\alpha > 0$
  - Factor: $e^{+\alpha} > 1$
  - Weight **increases**

Example: Weight Update Example

Suppose $\alpha = 0.5$ (a moderately good classifier):

- **Correctly classified**: Weight multiplied by $e^{-0.5} \approx 0.61$ (decreases by 39%)
- **Misclassified**: Weight multiplied by $e^{0.5} \approx 1.65$ (increases by 65%)

If $\alpha = 2.0$ (a very good classifier):

- **Correctly classified**: Weight multiplied by $e^{-2} \approx 0.14$ (decreases by 86%)
- **Misclassified**: Weight multiplied by $e^{2} \approx 7.4$ (increases by 640%!)

The better the classifier, the more dramatically we adjust weights!

4 How Weak Learners Use Weights

4.1 Option 1: Weighted Loss Function

When training the decision stump, instead of counting misclassifications, we compute the **weighted error**:

$$\text{Weighted Error} = \sum_{i:h(x_i)\neq y_i} w_i$$

For decision trees using Gini impurity:

$$\text{Weighted Gini} = \sum_i w_i \cdot \mathbf{1}[\text{sample } i \text{ in this node}] \cdot \text{impurity contribution}$$

This makes misclassifying high-weight samples more "costly," so the stump avoids those mistakes.

4.2 Option 2: Resampling

Alternatively, we can **resample** the training data according to the weights:

1. Generate a new training set by sampling with replacement
2. Probability of selecting sample i is proportional to w_i
3. Train a regular (unweighted) stump on this resampled data

This approach:

- Implicitly gives high-weight samples more influence
- Works with any base learner (doesn't need to support weights)
- Introduces additional randomness

5 Visual Example: 2D Classification

Example: AdaBoost in Action

Consider classifying two groups: orange circles and blue triangles.

Round 1:

- All 10 samples have weight $w = 0.1$
- First stump: "Is $x_1 > 4.6$?"
- Correctly classifies 7, misclassifies 3 (all orange circles)
- $\epsilon_1 = 0.1 + 0.1 + 0.1 = 0.3$
- $\alpha_1 = \frac{1}{2} \ln\left(\frac{0.7}{0.3}\right) \approx 0.42$

After Round 1 weight update:

- Correct samples: weights $\rightarrow 0.1 \times e^{-0.42} \approx 0.066$
- Misclassified samples: weights $\rightarrow 0.1 \times e^{0.42} \approx 0.152$
- After normalization: misclassified samples have $\sim$2.3x the weight

Round 2:

- Second stump must pay more attention to those 3 orange circles
- Chooses "Is $x_2 > 8$?" to separate them
- This might misclassify some previously correct blue triangles
- Process continues...

Final result:

- 3 simple axis-aligned splits
- Combined: complex decision boundary
- Can separate the groups better than any single stump!

6 AdaBoost vs. Gradient Boosting

Both are boosting methods, but they differ in key ways:

Table 2: *Comparison: AdaBoost vs. Gradient Boosting*

Aspect	AdaBoost	Gradient Boosting
Loss function	Exponential loss	Any differentiable loss
How it learns	Reweights samples	Fits to residuals/gradients
Learner weight α	Computed from formula	User-specified (learning rate)
Primary use	Classification	Classification and Regression
Sensitivity to outliers	High	Moderate
Overfitting tendency	Moderate	Can be controlled

6.1 The Connection: AdaBoost IS Gradient Boosting

Important: AdaBoost

AdaBoost can be viewed as a special case of gradient boosting where:

- Loss function: $L(y, f) = \exp(-y \cdot f)$ (exponential loss)
- This loss heavily penalizes confident wrong predictions

The weight update in AdaBoost is actually computing the gradient of exponential loss!

6.2 Exponential Loss

Definition: Exponential Loss

For labels $y \in \{-1, +1\}$ and prediction $f(x)$:

$$L(y, f) = \exp(-y \cdot f(x))$$

Properties:

- Correct confident prediction ($y \cdot f \gg 0$): Loss ≈ 0
- Wrong confident prediction ($y \cdot f \ll 0$): Loss $\to \infty$
- Serves as an upper bound on 0-1 classification error

7 Overfitting and Hyperparameters

7.1 AdaBoost Can Overfit!

Unlike some claims, boosting methods (including AdaBoost) **can overfit**:

Warning

Why AdaBoost Overfits:

- AdaBoost **obsesses** over misclassified samples
- Some "hard" samples might be outliers or noise
- With enough iterations, it memorizes these noise points
- Result: Perfect training accuracy, poor test performance

Solution: Use early stopping based on validation error!

7.2 Key Hyperparameters

Table 3: *AdaBoost Hyperparameters in sklearn*

Parameter	Default	Description
`n_estimators`	50	Number of weak learners
`learning_rate`	1.0	Shrinkage factor for α
`estimator`	DecisionTree(depth=1)	Base weak learner

Tuning advice:

- `n_estimators`: More = more complex. Use validation to find optimal.
- `learning_rate`: Lower values require more estimators but often generalize better.
- `estimator`: Stumps (depth 1) are standard; depth 2-3 can help but risk overfitting.

8 AdaBoost in Python

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# Prepare data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create AdaBoost classifier
# Note: sklearn uses SAMME algorithm (similar to original AdaBoost)
ada = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1),  # Stump
    n_estimators=50,
    learning_rate=1.0,
    algorithm='SAMME',
    random_state=42
)

# Train
ada.fit(X_train, y_train)

# Evaluate
print(f"Train Accuracy: {ada.score(X_train, y_train):.4f}")
print(f"Test Accuracy: {ada.score(X_test, y_test):.4f}")
```

8.1 Visualizing Training Progress

```python
# Plot training and test error vs. number of estimators
train_errors = []
test_errors = []

for n in range(1, 101):
    ada = AdaBoostClassifier(
        estimator=DecisionTreeClassifier(max_depth=1),
        n_estimators=n,
        random_state=42
    )
    ada.fit(X_train, y_train)
    train_errors.append(1 - ada.score(X_train, y_train))
    test_errors.append(1 - ada.score(X_test, y_test))

plt.figure(figsize=(10, 6))
plt.plot(range(1, 101), train_errors, label='Training Error')
plt.plot(range(1, 101), test_errors, label='Test Error')
plt.xlabel('Number of Estimators')
plt.ylabel('Error Rate')
```

```
plt.title('AdaBoost Learning Curve')
plt.legend()
plt.show()

# Find optimal number
best_n = np.argmin(test_errors) + 1
print(f"Optimal n_estimators: {best_n}")
```

8.2 Getting Feature Importance

```
# AdaBoost provides feature importance
importances = ada.feature_importances_

# Display
import pandas as pd
pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values('Importance', ascending=False)
```

9 When to Use AdaBoost

9.1 Advantages

- **Simple to implement**: Algorithm is straightforward
- **No hyperparameter for α**: Learner weights computed automatically
- **Works with any weak learner**: Not limited to trees
- **Feature importance**: Built-in interpretation
- **Historical importance**: Foundation of modern boosting

9.2 Disadvantages

- **Sensitive to outliers**: Keeps increasing weights on hard-to-classify points
- **Sensitive to noise**: Treats noisy samples as "important"
- **Can overfit**: Especially with many iterations
- **Limited to classification**: Original form; Gradient Boosting is more flexible

9.3 Modern Alternatives

In practice, you'll often use:

- **XGBoost**: Optimized gradient boosting with regularization
- **LightGBM**: Fast gradient boosting for large datasets
- **CatBoost**: Handles categorical features well

These are all variants of Gradient Boosting rather than AdaBoost, but understanding AdaBoost helps you understand the foundations.

10 Key Takeaways

Key Summary

AdaBoost Core Ideas:

- Combines weak learners (stumps) into a strong classifier
- Uses **sample weights** instead of residuals
- Misclassified samples get higher weights
- Better classifiers get more voting power (α)

The Algorithm:

1. Initialize all sample weights equally: $w_i = 1/N$
2. For each iteration:
   - Train stump on weighted data
   - Compute weighted error ϵ
   - Compute learner weight $\alpha = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$
   - Update sample weights: $w_i \leftarrow w_i \cdot \exp(-\alpha y_i h(x_i))$
3. Final prediction: weighted vote of all stumps

Key Formulas:

- Learner weight: $\alpha = \frac{1}{2} \ln \left(\frac{1-\epsilon}{\epsilon} \right)$
- Sample weight update: $w_{new} \propto w_{old} \cdot \exp(-\alpha \cdot y \cdot \hat{y})$
- Final prediction: $\text{sign} \left(\sum_m \alpha_m h_m(x) \right)$

Practical Tips:

- Use validation data to avoid overfitting
- Monitor both training and test error
- Consider XGBoost/LightGBM for production use

11 Practice Questions

1. **Conceptual**: Explain why AdaBoost uses labels $\{-1, +1\}$ instead of $\{0, 1\}$. How does this simplify the weight update formula?
2. **Calculation**: A weak learner has weighted error $\epsilon = 0.2$. Calculate α and explain what this value means for the learner's influence.
3. **Weight Update**: If a sample has current weight $w = 0.1$ and $\alpha = 0.5$, what is the new weight if:
   - The sample is correctly classified?
   - The sample is misclassified?
4. **Edge Cases**: What happens in AdaBoost if a weak learner achieves:
   - Perfect accuracy ($\epsilon = 0$)?
   - Random guessing ($\epsilon = 0.5$)?
   - Worse than random ($\epsilon = 0.7$)?
5. **Comparison**: Explain the key difference between how AdaBoost and Gradient Boosting "learn from mistakes."
6. **Overfitting**: Why can AdaBoost overfit even though each weak learner is simple? How would you detect and prevent this?
7. **Code**: Modify the sklearn AdaBoost example to:
   - Use depth-2 trees instead of stumps
   - Implement early stopping based on validation error
   - Compare performance to the default configuration

■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 25: Blending, Stacking, and Mixture of Experts
■ **Instructors:**	Pavlos Protopapas, Kevin Rader, Chris Gumb
■ **Objective:**	Learn advanced ensemble methods that combine heterogeneous models, understand meta-learning, and explore the Mixture of Experts architecture used in modern LLMs

Contents

1 The Big Picture: Combining Different Models

Key Summary

This final lecture introduces **advanced ensemble methods** that go beyond Bagging and Boosting:

- **Previous methods** (Bagging, Random Forest, Boosting): Combine **homogeneous** models (same type, e.g., all decision trees)
- **New methods** (Blending, Stacking, MoE): Combine **heterogeneous** models (different types, e.g., logistic regression + random forest + KNN)

Key insight: A **meta-model** learns how to best combine predictions from diverse base models!

1.1 Why This Matters

Key Information

Practical Relevance:

- In data science projects, you often try multiple models (logistic regression, SVM, random forest, boosting, etc.)
- Instead of choosing the "best" one, why not combine them?
- This is extremely valuable for final projects and Kaggle competitions
- **Modern LLMs** (like DeepSeek, Mistral) use Mixture of Experts for efficiency—this architecture is revolutionizing AI!

1.2 Review: What We've Learned

Table 1: *Ensemble Methods Review*

Method	Base Models	Training	Aggregation	Goal
Bagging	Homogeneous (trees)	Parallel	Average/Vote	Reduce variance
Random Forest	Homogeneous (trees)	Parallel	Average/Vote	Reduce variance
Boosting	Homogeneous (stumps)	Sequential	Weighted sum	Reduce bias
Blending	**Heterogeneous**	Parallel	**Meta-model**	Best of all
Stacking	**Heterogeneous**	Parallel + CV	**Meta-model**	Best of all

2 Blending: Simple Heterogeneous Ensembles

Definition: Blending

Blending is an ensemble method that:

1. Trains multiple **different** base models on training data
2. Gets predictions from these models on a held-out validation set
3. Trains a **meta-model** to learn how to combine these predictions

The meta-model learns "when to trust which model."

2.1 The Blending Architecture

Data → **Base Models** (Logistic, RF, KNN, ...) → **Predictions** → **Meta-Model** → **Final Prediction**

2.2 Step-by-Step Blending Process

Step 1: Split the Data

Split your data into four parts:

- **Training set**: To train base models
- **Validation set**: To generate predictions for meta-model training
- **Hold-out set**: To validate the meta-model
- **Test set**: Final evaluation (untouched until the end!)

Step 2: Train Base Models

Train multiple different models on the training set:

- Model 1: Logistic Regression
- Model 2: Random Forest
- Model 3: Gradient Boosting
- Model 4: KNN
- ... (as many as you want!)

Step 3: Generate Validation Predictions

Use each trained base model to predict on the validation set. This gives you:

- $\hat{y}_1$: Predictions from Model 1
- $\hat{y}_2$: Predictions from Model 2
- ... and so on

Step 4: Create Meta-Model Training Data

Combine the predictions into a new dataset:

$$X_{meta} = [\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4, (\text{optionally: original X})]$$

$$y_{meta} = \text{true labels from validation set}$$

Step 5: Train the Meta-Model

Train a simple model (often linear regression or logistic regression) on this meta-dataset:

$$\text{Final prediction} = f_{meta}(X_{meta})$$

Step 6: Evaluate on Hold-out and Test

Use the same process for hold-out (to tune meta-model) and test (final evaluation).

2.3 Why Use a Simple Meta-Model?

Key Information

Recommendation: Keep the Meta-Model Simple!

A simple meta-model (like linear regression) has advantages:

- **Interpretability**: Coefficients tell you which base model is most trusted
- **Avoids overfitting**: Complex meta-models can overfit to validation predictions
- **Fast to train**: Meta-model training should be quick

For example, if the meta-model learns:

$$\hat{y}_{final} = 0.5 \cdot \hat{y}_{RF} + 0.3 \cdot \hat{y}_{GB} + 0.2 \cdot \hat{y}_{LR}$$

You know Random Forest contributes most to the final prediction!

2.4 The Passthrough Option

Definition: Passthrough

Passthrough means including the original features X along with the base model predictions when training the meta-model.

Without passthrough: $X_{meta} = [\hat{y}_1, \hat{y}_2, ..., \hat{y}_K]$

With passthrough: $X_{meta} = [X, \hat{y}_1, \hat{y}_2, ..., \hat{y}_K]$

Why use passthrough?

- The meta-model can learn "when input looks like THIS, trust Model 2 more"
- Base models might miss some signal that the meta-model can capture
- Generally improves performance

Warning

In sklearn, `passthrough=False` by default. The instructor recommends setting `passthrough=True` for better results!

3 Stacking: Blending with Cross-Validation

Definition: Stacking

Stacking (Stacked Generalization) is similar to blending but uses **cross-validation** instead of a fixed validation split. This makes better use of data!

3.1 The Problem with Blending

Blending requires:

- Training set (for base models)
- Validation set (for meta-model training)
- Hold-out set (for meta-model validation)
- Test set (for final evaluation)

That's 4 separate datasets! With limited data, this is wasteful.

3.2 Stacking Solution: Out-of-Fold Predictions

Stacking uses K-fold cross-validation to generate "out-of-fold" predictions:

1. Split training data into K folds (e.g., K=5)
2. For each fold:
   - Train base models on K-1 folds
   - Predict on the held-out fold
3. After K iterations, every sample has a prediction from a model that **didn't see it during training**
4. These predictions become the meta-model's training data

Example: 3-Fold Stacking Example

Data: 900 samples, 3 folds of 300 each

Round 1:

- Train on Folds 2+3 (600 samples)
- Predict on Fold 1 (300 samples) $\rightarrow \hat{y}_1^{(1)}$

Round 2:

- Train on Folds 1+3 (600 samples)
- Predict on Fold 2 (300 samples) $\rightarrow \hat{y}_1^{(2)}$

Round 3:

- Train on Folds 1+2 (600 samples)
- Predict on Fold 3 (300 samples) $\rightarrow \hat{y}_1^{(3)}$

Result: 900 out-of-fold predictions for Model 1!

Repeat for all base models. Then train meta-model on these predictions.

3.3 Stacking vs. Blending

Table 2: *Stacking vs. Blending Comparison*

Aspect	**Blending**	**Stacking**
Data efficiency	Lower	Higher
Implementation	Simpler	More complex
Computation	Faster	Slower (K-fold)
Risk of overfitting	Higher	Lower
Recommended for	Large datasets	Smaller datasets

4 Stacking in Python

```python
from sklearn.ensemble import StackingClassifier, StackingRegressor
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Define base models (heterogeneous!)
estimators = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('gb', GradientBoostingClassifier(n_estimators=100, random_state=42)),
    ('knn', KNeighborsClassifier(n_neighbors=5))
]

# Define meta-model (keep it simple!)
meta_model = LogisticRegression()

# Create stacking classifier
stacking_clf = StackingClassifier(
    estimators=estimators,
    final_estimator=meta_model,
    cv=5,                    # 5-fold cross-validation
    passthrough=True,        # Include original features!
    stack_method='predict_proba'  # Use probabilities, not labels
)

# Train
stacking_clf.fit(X_train, y_train)

# Evaluate
print(f"Stacking Train Accuracy: {stacking_clf.score(X_train, y_train):.4f}")
print(f"Stacking Test Accuracy: {stacking_clf.score(X_test, y_test):.4f}")
```

4.1 Important: Use Probabilities, Not Labels

Warning

When stacking classifiers, use `stack_method='predict_proba'` instead of hard labels!

Why?

- Labels are discrete (0 or 1)—limited information
- Probabilities are continuous (0.0 to 1.0)—much richer signal
- Meta-model can learn "Model A is confident, Model B is uncertain"

5 Mixture of Experts (MoE)

Key Summary

Mixture of Experts takes a fundamentally different approach:
Instead of combining all models' outputs, MoE **selects which expert(s) to use** based on the input!
This is the architecture behind modern efficient LLMs like DeepSeek and Mistral.

5.1 The Core Idea

In previous ensembles, ALL models contribute to every prediction:

$$\hat{y} = \frac{1}{K}\sum_{k=1}^{K} f_k(x) \quad \text{(everyone votes)}$$

In MoE, a **gating network** decides which experts to activate:

$$\hat{y} = \sum_{k=1}^{K} g_k(x) \cdot f_k(x) \quad \text{(specialists handle their domain)}$$

where $g_k(x)$ is the "gate" for expert k, and gates sum to 1: $\sum_k g_k(x) = 1$

5.2 Components of MoE

Definition: Mixture of Experts Components

1. **Experts** (f_k): Individual models that specialize in different regions of the input space
2. **Gating Network** (g): A model that looks at the input x and outputs weights for each expert
3. **Combination**: Final prediction is a weighted combination based on gates

Example: Medical Diagnosis MoE

Experts:

- Expert 1: Radiologist (specializes in imaging)
- Expert 2: Pathologist (specializes in lab tests)
- Expert 3: General Practitioner (handles common cases)

Gating Network: Looks at patient symptoms and test results

For a patient with MRI scan:

- Gate outputs: $g_1 = 0.7$, $g_2 = 0.2$, $g_3 = 0.1$
- Radiologist's opinion is weighted most heavily!

For a patient with blood work:

- Gate outputs: $g_1 = 0.1$, $g_2 = 0.8$, $g_3 = 0.1$
- Pathologist's opinion dominates!

5.3 The Gating Network

The gating network uses **softmax** to ensure gates sum to 1:

$$g_k(x) = \frac{\exp(\alpha_k^T x)}{\sum_{j=1}^{K} \exp(\alpha_j^T x)}$$

where α_k are learnable parameters for expert k.

Key Information

This is exactly like **multinomial logistic regression** (softmax)! The gating network is essentially classifying "which expert should handle this input."

5.4 Training MoE: The Challenge

Important: Expert Collapse Problem

A naive loss function can cause **expert collapse**:

- One expert becomes dominant
- Other experts never get trained
- System degenerates to single-model performance

This is what happened with early implementations (like Mistral's initial attempts).

Solution: Use a loss function that gives each expert its own error signal, not just the global error.

5.4.1 Bad Loss (Causes Collapse)

$$L = \sum_n (y_n - \sum_k g_k(x_n) \cdot f_k(x_n))^2$$

Problem: All experts see the same global error. They all try to do the same thing and collapse.

5.4.2 Good Loss (Prevents Collapse)

$$L = \sum_n \sum_k g_k(x_n) \cdot (y_n - f_k(x_n))^2$$

Benefit: Each expert gets weighted error specific to its domain. Experts specialize!

5.5 Why MoE Matters for LLMs

Key Information

Sparse MoE in Large Language Models:

Modern LLMs like DeepSeek use MoE to be more efficient:

- Model has billions of parameters spread across many experts
- For each input, only a **few experts** are activated (sparse activation)

- This gives the **capacity** of a huge model with the **computation** of a small one

Example: A model with 100 experts but only top-2 activated per token can have 10x the parameters with only 2x the compute!

6 Practical Guidelines

6.1 When to Use Each Method

Table 3: *Choosing an Ensemble Method*

Situation	Recommended Method
Quick baseline, plenty of data	Random Forest or Gradient Boosting
Want to squeeze last 1% accuracy	Stacking with diverse models
Data has distinct regions/modes	Mixture of Experts
Need interpretability	Stacking with linear meta-model
Limited computational resources	Blending (simpler than stacking)
Building a large-scale system	Sparse Mixture of Experts

6.2 Best Practices

1. **Diversity is key**: Use models with different "philosophies"
   - Linear model (captures global trends)
   - Tree model (captures interactions)
   - KNN (captures local patterns)
   - Boosting (captures complex patterns)
2. **Watch for data leakage**: Never use test data during stacking/blending
3. **Keep meta-model simple**: Complex meta-models tend to overfit
4. **Use probabilities for classification**: More informative than labels
5. **Enable passthrough**: Let meta-model see original features
6. **Validate properly**: Use hold-out set for meta-model tuning

7 Course Summary: The ML Toolkit

Key Summary

What You've Learned in CS109A:

Regression:

- Linear Regression, Polynomial Regression
- Regularization (Ridge, Lasso)

Classification:

- Logistic Regression
- K-Nearest Neighbors

Probabilistic Methods:

- Bayesian Inference
- MCMC Sampling

Tree-Based Methods:

- Decision Trees (Classification and Regression)
- Bagging and Random Forests
- Gradient Boosting and AdaBoost

Advanced Ensembles:

- Blending and Stacking
- Mixture of Experts

Important Concepts Throughout:

- Bias-Variance Tradeoff
- Cross-Validation
- Feature Engineering
- Model Selection and Evaluation

Key Information

Final Thoughts from the Instructors:

1. **Use office hours!** You learn more in 30 minutes with a TA than hours struggling alone.
2. **Don't over-rely on LLMs**: They're good for small questions, but can lead you astray for learning core concepts.
3. **Start simple**: Even with all these methods, linear/logistic regression should be your first attempt. Understand the basics before reaching for complex tools.
4. **For tabular data**: Random Forest and Gradient Boosting often beat deep learning!
5. **Keep learning**: This course is just the beginning. There's much more in CS109B and beyond.

8 Practice Questions

1. **Conceptual**: Explain the difference between homogeneous and heterogeneous ensembles. Give an example of each.
2. **Blending**: Why do we need a separate validation set for blending? What would happen if we used the same data to train base models and generate meta-model training data?
3. **Stacking**: Explain what "out-of-fold predictions" are and why they prevent overfitting in the meta-model.
4. **Passthrough**: What is the passthrough option in stacking? Why is it recommended to enable it?
5. **MoE Gating**: In a Mixture of Experts model, what does the gating network do? What function ensures the gate outputs sum to 1?
6. **Expert Collapse**: What is expert collapse in MoE, and why does a naive loss function cause it?
7. **Practical**: You're competing in a Kaggle competition. You've trained:
   - Logistic Regression: 82% accuracy
   - Random Forest: 85% accuracy
   - Gradient Boosting: 86% accuracy
   - KNN: 80% accuracy

   How would you use stacking to potentially improve beyond 86%? Write the sklearn code.
8. **LLM Application**: Why are Large Language Models using Mixture of Experts architectures? What efficiency benefit does sparse MoE provide?