Building Recommender Systems (DRAFT)

Tri Kurniawan Wijaya

April 2, 2025

Disclaimer

Preface

If you are picking up this book, chances are you are already intrigued by how recommender systems shape our digital experiences, from the movies we watch to the products we discover online. And to be honest, I am too.

My name is Tri, and I am a user of recommender systems every single day. But beyond being a consumer, I have also had the privilege of being a creator. Over the years, I have worked in both academia and industry, designing, building, and researching recommender systems. I have led teams dedicated to crafting them, and I have even contributed to the academic literature in this field. This book stems from that dual perspective – as both a practitioner in the trenches and a curious explorer of the underlying principles.

Why write a book about recommender systems? Imagine a world without them for a moment. Navigating the vast ocean of information and options online would feel... well, cumbersome, to say the least. Recommender systems, at their best, are like digital guides, helping us discover what truly resonates with us amidst the overwhelming noise. They enhance our experiences, broaden our horizons, and, when done well, can feel almost magical.

However, The path from theory to implementation in recommender systems is not always a smooth, perfectly optimized one. In the real world, things can, and often do, get a little... derailed. Building effective recommender systems is complex. Data is messy. User behavior is unpredictable. And, importantly, the goals of the organizations deploying these systems are not always perfectly aligned with the user's ideal experience.

Companies that create and deploy recommender systems have a business to run. Revenue, profitability, and sustainability are essential for their existence. Sometimes, pursuing these business goals means trade-offs. There can be a delicate balance – or sometimes a tension – between prioritizing user satisfaction and focusing on short-term revenue objectives. Ideally, these priorities align, creating a win-win scenario – a "rainbow world" as I like to think of it, where user delight and business success go hand in hand. But reality is often more nuanced. When these priorities diverge, users can feel it. We have all experienced recommendations that miss the mark, or systems that seem to prioritize engagement metrics over genuine helpfulness. Customer complaints are a natural consequence of this sometimes imperfect alignment. Yet, we also need to remember that without these companies striving for viability, those services, and indeed those recommender systems, would not exist in the first place.

This book aims to navigate this exciting and sometimes contradictory landscape. We will explore the foundational principles and cutting-edge techniques that make recommender systems work. We will delve into the algorithms, the data, and increasingly, the powerful language models that are reshaping the field. But we will also keep a practitioner's eye on the real-world challenges and trade-offs.

The field of recommender systems is in constant motion. New research emerges rapidly, new models are developed, and best practices are continually being refined. Consider this book as a snapshot in time, a foundation to build upon, and a guide to help you navigate this ever-evolving domain. My hope is that by the end of this book, you'll not only understand the "how" of recommender systems, but also the "why" and the "what-ifs" – the magic, the messiness, and ultimately, the immense potential of this field to shape our digital future. More than that, I hope it empowers you to put this knowledge into practice, to craft one of your own.

Tri -

Contents

Part I

Foundations and Introduction to Recommender Systems

Chapter 1

Introduction to Recommender Systems

Make it easier to find what matters.
— A possible distillation of the purpose of Recommender Systems

1.1 What are Recommender Systems?

Imagine a world without recommender systems. Think about browsing an online store where products are displayed in no particular order – perhaps just alphabetically, or completely randomly. Or envision a music streaming service that simply plays songs from its entire library on shuffle, with no attempt to personalize the selection to your taste. In such a world, finding movies, songs, or products you might actually like would be incredibly challenging, wouldn't it? Unless you knew exactly what you were looking for beforehand and could search directly, discovery would be a cumbersome and often frustrating process.

But what if there was an automatic system, an intelligent guide, that could present you with choices specifically tailored to your interests? A system that proactively surfaces movies, songs, articles, or products that you are likely to enjoy, often even before you explicitly search for them. This is precisely the promise of *recommender systems*. They are designed to bring order to the chaos of overwhelming choice, and to connect you with items that are personally relevant and appealing.

At their core, recommender systems are intelligent software tools and techniques designed to *predict user preferences and suggest items that users might find interesting, relevant, or useful*. They act as personalized filters in the vast digital landscape, guiding us towards the most appealing options based on our past behaviors, expressed preferences, and contextual information.

Let's consider some everyday examples:

- *Netflix suggesting movies and TV shows:* Based on your viewing history, ratings, and what similar users have enjoyed, Netflix recommends content tailored to your taste.
- *Amazon recommending products:* When you browse Amazon, you see "Customers who bought this item also bought...", "Recommended for you...", and personalized product suggestions throughout the site, all powered by recommender systems.
- *YouTube suggesting videos:* The "Up Next" and "Recommended for you" sections on YouTube are driven by algorithms that analyze your watch history, subscriptions, and trending content to keep you engaged.
- *Spotify and Apple Music recommending songs and playlists:* These music streaming services create personalized playlists like "Discover Weekly" or "New Music Mix" based on your listening habits and preferences.
- *News websites suggesting articles:* Many news outlets personalize article recommendations to keep you informed on topics you are interested in.

- *Social media platforms suggesting people to follow and content to engage with:* Platforms like Twitter, LinkedIn, TikTok, and Facebook use recommender systems to suggest connections, groups, and posts that might be relevant to your interests and network.

These are just a few examples, and recommender systems are increasingly becoming an invisible yet vital part of our digital lives. They operate behind the scenes in countless applications, shaping our online experiences and influencing our decisions.

1.2 Why Recommender Systems Matter?

The widespread adoption of recommender systems is not accidental. They offer significant benefits to both users and businesses, making them a crucial technology in today's information-rich environment.

1.2.1 User Experience and Personalization

From a user's perspective, recommender systems offer:

- *Reduced information overload and choice paralysis.* In a world of overwhelming choices, recommender systems act as filters, helping users navigate vast catalogs of items and quickly find what they are looking for or what they might enjoy. This reduces the cognitive burden and frustration of endless browsing.
- *Personalized and relevant experiences.* Recommender systems deliver content and suggestions tailored to individual tastes and needs. This creates a more personalized and satisfying online experience, making users feel understood and valued.
- *Discovery of new and interesting items.* Beyond simply finding what users already know they want, recommender systems can also introduce them to new items they might not have discovered otherwise, expanding their horizons and potentially leading to serendipitous discoveries. Think of discovering a new genre of music or a hidden gem of a movie through recommendations.
- *Increased efficiency and convenience.* Recommender systems save users time and effort by proactively suggesting relevant items. Users spend less time searching and more time enjoying content or making informed decisions.

1.2.2 Business Value

For businesses, recommender systems are powerful tools to drive:

- *Increased sales and revenue.* By suggesting relevant products or services, recommender systems can significantly boost sales. Personalized recommendations encourage users to discover items they might not have found otherwise, leading to increased purchases and higher average order values. Think of Amazon's product recommendations which are estimated to contribute significantly to their overall revenue.
- *Improved customer engagement and retention.* Personalized experiences are more engaging. Recommender systems help keep users interested and active on platforms by consistently suggesting content that resonates with them. Increased engagement translates to longer user sessions, more frequent visits, and higher customer retention rates.
- *Enhanced customer lifetime value.* By fostering loyalty and encouraging repeat business, recommender systems contribute to a higher customer lifetime value. Satisfied users are more likely to remain customers long-term and become advocates for the brand.
- *Effective marketing and promotion.* Recommender systems allow for targeted and personalized marketing campaigns. Businesses can promote specific products or services to users who are most likely to be interested, improving marketing efficiency and return on investment.

- *Data insights and understanding customer behavior.* The data collected by recommender systems (user interactions, preferences) provides valuable insights into customer behavior, preferences, and trends. This information can be used to improve product development, marketing strategies, and overall business decisions.

Recommender systems strive to create a *win-win* situation, where businesses benefit from increased revenue and engagement, while users enjoy personalized, efficient, and enriching online experiences. This symbiotic relationship has fueled the growth and importance of recommender systems in the digital age.

1.3 The Recommendation Problem Definition

To understand how recommender systems work, it is helpful to formalize the recommendation problem. At a high level, we can think of it as follows:

We have a set of *users* $U = \{u_1, u_2, ..., u_m\}$ and a set of *items* $I = \{i_1, i_2, ..., i_n\}$. Users could be customers, viewers, readers, listeners, etc., and items could be products, movies, articles, songs, etc.

Our goal is to build a system that, for a given user $u \in U$, can recommend a ranked list of items $R_u \subseteq I$ that the user u will likely find interesting or valuable.

To achieve this, recommender systems typically rely on data about user-item interactions and item characteristics. The input data can be broadly categorized into:

- **User-item interaction.**
  - *Explicit feedback:* ratings (e.g., 1-5 stars), reviews, likes/dislikes, direct preferences expressed by users. This is often sparse and not always available.
  - *Implicit feedback:* purchase history, browsing history, clicks, watch time, listening frequency, social interactions (follows, shares). This is often more readily available and can be inferred from user behavior.
- **User features or profiles.**
  - *Demographic information:* age, gender, location, demographics (if available and ethically permissible).
  - *User preferences and interests:* explicitly stated interests, derived interests from behavior, topics they follow, categories they browse.
- **Item features or content.**
  - *Metadata:* category, genre, author, actors, director, brand, price, specifications, etc.
  - *Content descriptions:* textual descriptions, summaries, reviews, tags, images, audio, video content itself.
- **Contextual information.**
  - *Time:* time of day, day of week, season.
  - *Location:* user's geographical location.
  - *Device:* mobile, desktop, tablet.
  - *Social context:* user's social network, friends' preferences.
  - *Session information:* current browsing session, recent interactions.

The core task of building a recommender system is to design algorithms that can effectively process this input data and generate accurate and relevant recommendations as output. The choice of algorithm and data to use depends heavily on the specific application, data availability, and desired recommendation goals. The *output* of a recommender system is typically:

- *A ranked list of items.* A list of items ordered by predicted relevance or preference for a given user. This is the most common output format, used for displaying recommendations on websites and apps.
- *A prediction score.* A numerical score representing the predicted preference, or rating of a specific user for a specific item. This can be used for filtering, ranking, or as input to other systems.
- *A set of top-N recommendations.* The top N most relevant items for a user.

1.4 Types of Recommender Systems

Before diving into the technical details, it is helpful to briefly introduce the main categories of recommender systems we will explore in this book. These categories are not always mutually exclusive, and real-world systems often combine elements from multiple approaches, but they provide a useful framework for understanding the fundamental methodologies:

- **Content-based recommender systems.** These systems recommend items that are similar to items a user has liked in the past, *based on item content and features*. They focus on the characteristics of items and user preferences for those characteristics. For example, recommending movies of the same genre or with the same actors as movies a user has previously enjoyed.
- **Collaborative filtering (CF) recommender systems.** CF systems make recommendations based on the *past interactions* and *preferences of a community of users*. They leverage the idea that users who have agreed in the past tend to agree again in the future. There are two main types:
  - *User-based collaborative filtering:* recommends items that *users similar to the target user* have liked.
  - *Item-based collaborative filtering:* recommends items that are *similar to items* the target user has liked, based on user interaction patterns.
- **Knowledge-based recommender systems.** These systems rely on *explicit knowledge* about items and user needs and preferences. They often use rule-based approaches, constraint satisfaction, or case-based reasoning to generate recommendations, especially useful in domains where content is sparse or user preferences are complex. Examples include recommending products based on specified features or helping users configure complex products.
- **Hybrid recommender systems.** As the name suggests, these systems *combine two or more* different recommendation techniques to leverage their strengths and mitigate their weaknesses. Hybrid approaches are common in practice to achieve better overall performance and address limitations of individual methods.
- **Deep learning-based recommender systems.** This is a more recent and rapidly evolving category that utilizes the power of *deep neural networks* to learn complex user and item representations, model user-item interactions, and generate sophisticated recommendations. Deep learning is increasingly being applied to enhance all types of recommender systems.

We will delve into each of these categories in subsequent chapters, exploring their algorithms, advantages, disadvantages, and practical applications.

1.5 Applications of Recommender Systems

Recommender systems are not confined to just a few industries; they are pervasive across a wide range of domains, transforming how we interact with information and services. Here are some prominent application areas:

- *E-commerce.* Product recommendations on e-commerce websites (Amazon, e-Bay, etc.) are perhaps the most well-known application. They drive sales, improve product discovery, and personalize the shopping experience.

- *Streaming services (movies, music, video).* Netflix, Spotify, YouTube, and similar platforms heavily rely on recommender systems to suggest movies, TV shows, music, and videos that keep users engaged and subscribed.
- *Social media.* Many platforms use recommendations to suggest friends to connect with, groups to join, content to follow, and posts to engage with, shaping user experiences and platform growth.
- *News and content aggregation.* News websites, news aggregators, and content discovery platforms use recommender systems to personalize news feeds, suggest articles, and keep users informed on relevant topics.
- *Travel and hospitality.* Recommender systems in travel websites and apps suggest hotels, flights, restaurants, attractions, and travel packages based on user preferences and travel history.
- *Advertising.* Personalized advertising relies heavily on recommender systems to target ads to users who are most likely to be interested in the advertised products or services.
- *Personalized search.* Recommender systems can enhance search engines by personalizing search results based on user context and past behavior.
- *Education and online learning.* Recommender systems can personalize learning paths, suggest relevant courses, learning materials, and educational resources to individual students.
- *Job search and recruitment.* Platforms like LinkedIn and job boards use recommender systems to suggest relevant job openings to job seekers and recommend candidates to recruiters.

This diverse range of applications underscores the broad applicability and transformative potential of recommender systems in various aspects of modern life.

1.6 Challenges in Building Recommender Systems

While recommender systems offer significant benefits, building effective and robust systems is not without its challenges. We will encounter many of these challenges throughout this book. Some key challenges include:

- *Data sparsity.* User-item interaction data is often sparse. Users typically interact with only a small fraction of all available items. This sparsity makes it challenging to accurately infer user preferences and item similarities.
- *Cold start problem.* How we can provide recommendations for new users (user cold start) or new items (item cold start) that have little to no interaction history.
- *Scalability.* Recommender systems often need to handle massive datasets with millions of users and items and generate recommendations in real-time or near real-time. Scalability and efficiency are crucial.
- *Serendipity and novelty.* Going beyond simply recommending popular or obvious items. Users often appreciate recommendations that are surprising, novel, and help them discover items they wouldn't have found on their own.
- *Diversity and filter bubbles.* Recommender systems can inadvertently create filter bubbles by over-personalizing recommendations and limiting users' exposure to diverse perspectives and items. Promoting diversity in recommendations is important.
- *Explainability and transparency.* Understanding *why* a recommender system makes a particular suggestion is increasingly important for user trust, debugging, and improving system design. Explainable recommendations are becoming a key focus.

- *Privacy and ethical considerations.* Recommender systems rely on user data, raising important ethical and privacy concerns. Building systems that are both effective and responsible is paramount.
- *Dynamic user preferences.* User tastes and preferences are not static; they evolve over time. Recommender systems need to adapt to these changing preferences and provide dynamic, up-to-date recommendations.
- *Evaluation complexity.* Evaluating recommender systems beyond simple accuracy metrics is challenging. Capturing user satisfaction, long-term engagement, and other qualitative aspects requires sophisticated evaluation methodologies.

Addressing these challenges is an active area of research and development in the field of recommender systems, and we will explore various techniques and strategies to tackle them in the chapters to come.

1.7 Book Roadmap

This book is structured to guide you through the journey of building effective recommender systems, from foundational concepts to advanced techniques and practical considerations. Here is a brief roadmap:

- **Part I (Chapters 1-2): Foundations and Introduction.** We have laid the groundwork in this chapter by defining recommender systems, understanding their importance, and formalizing the recommendation problem. Chapter 2 will equip you with the necessary mathematical and technical background.
- **Part II (Chapters 3-6): Core Recommender System Techniques.** We will delve into the fundamental algorithms: Content-Based Filtering, Collaborative Filtering (User-Based, Item-Based), and Matrix Factorization. You will learn the principles, algorithms, advantages, and disadvantages of each approach.
- **Part III (Chapters 7-9): Advanced Recommender System Techniques and Hybrid Approaches.** We will explore more sophisticated techniques, including Hybrid Recommender Systems, Knowledge-Based Recommender Systems, and the application of Deep Learning in this domain.
- **Part IV (Chapters 10-12): Evaluation, Practical Considerations, and Advanced Topics.** We will discuss robust evaluation methodologies, practical aspects of building real-world systems, and explore advanced topics like explainability, fairness, and privacy.
- **Part V (Chapters 13-14): Case Studies and Future Directions.** We will examine real-world case studies to learn from successful implementations and conclude with a look at future trends and open research challenges in the field.

By the end of this book, I hope you will have a solid understanding of the principles, techniques, and practical considerations necessary to design, build, and evaluate your own recommender systems.

Chapter 2

Mathematical and Technical Preliminaries

The laws of nature are written in the language of mathematics.
— Galileo Galilei

This chapter provides a review of essential mathematical and technical concepts that will be frequently used throughout this book. While I will strive to explain techniques in an accessible way, a basic understanding of these preliminaries will greatly enhance your comprehension of the algorithms and evaluation methodologies discussed in later chapters. This chapter is intended to be a refresher, not an in-depth course on each topic. If you are already familiar with these concepts, you may choose to skim or skip this chapter.

2.1 Linear Algebra Fundamentals

Linear algebra provides the mathematical language and tools for working with vectors and matrices, which are fundamental representations in many recommender system algorithms, especially *collaborative filtering* and *matrix factorization*.

2.1.1 Vectors and Matrices

- **Vectors.** A vector is a one-dimensional array of numbers, often representing features of a user or an item.

  Example: Consider representing a user's preferences for movie genres. A user vector $\mathbf{u}_1$ might be:

$$\mathbf{u}_1 = \begin{pmatrix} \text{Action} \\ \text{Comedy} \\ \text{Drama} \\ \text{Sci-Fi} \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 2 \\ 3 \end{pmatrix}$$

  Here, the vector $\mathbf{u}_1$ represents user 1's ratings (on a scale of 1-5) for different movie genres: Action (5 stars), Comedy (4 stars), Drama (2 stars), and Sci-Fi (3 stars).

- **Matrices.** A matrix is a two-dimensional array of numbers. In recommender systems, matrices are frequently used to represent user-item interaction data, such as the *user-item rating matrix* R, where R_{ui} represents the rating given by user u to item i.

Example: Consider a user-item rating matrix $\mathbf{R}$ for 3 users and 4 movies (Movie A, Movie B, Movie C, Movie D):

$$\mathbf{R} = \begin{pmatrix} & \text{Movie A} & \text{Movie B} & \text{Movie C} & \text{Movie D} \\ \text{User 1} & 5 & ? & 3 & 4 \\ \text{User 2} & ? & 4 & 5 & ? \\ \text{User 3} & 2 & 3 & ? & 5 \end{pmatrix} = \begin{pmatrix} 5 & ? & 3 & 4 \\ ? & 4 & 5 & ? \\ 2 & 3 & ? & 5 \end{pmatrix}$$

Here, $R_{11} = 5$ means User 1 rated Movie A as 5 stars. '?' indicates a missing rating (the user has not rated that movie).

- **Matrix Transpose.** The transpose of a matrix $\mathbf{M}$, denoted as $\mathbf{M}^T$, is obtained by interchanging its rows and columns.

  Example: Transposing the user-item rating matrix $\mathbf{R}$ from above:

$$\mathbf{R}^T = \begin{pmatrix} & \text{User 1} & \text{User 2} & \text{User 3} \\ \text{Movie A} & 5 & ? & 2 \\ \text{Movie B} & ? & 4 & 3 \\ \text{Movie C} & 3 & 5 & ? \\ \text{Movie D} & 4 & ? & 5 \end{pmatrix} = \begin{pmatrix} 5 & ? & 2 \\ ? & 4 & 3 \\ 3 & 5 & ? \\ 4 & ? & 5 \end{pmatrix}$$

  The rows of $\mathbf{R}$ become the columns of $\mathbf{R}^T$, and vice-versa.

2.1.2 Basic Matrix Operations

Matrix Addition and Subtraction

Matrices of the same dimensions can be added or subtracted element-wise.

Example (vector addition): Suppose we have two user preference vectors for genres:

$$\mathbf{u}_1 = \begin{pmatrix} 5 \\ 4 \\ 2 \\ 3 \end{pmatrix} \text{ (User 1 preferences)}$$

$$\mathbf{u}_2 = \begin{pmatrix} 3 \\ 5 \\ 4 \\ 1 \end{pmatrix} \text{ (User 2 preferences)}$$

We can calculate the average preference vector by adding them and dividing by 2 (scalar multiplication, see below, after addition):

$$\mathbf{u}_{avg} = \frac{1}{2}(\mathbf{u}_1 + \mathbf{u}_2) = \frac{1}{2}\left(\begin{pmatrix} 5 \\ 4 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 3 \\ 5 \\ 4 \\ 1 \end{pmatrix}\right) = \frac{1}{2}\begin{pmatrix} 8 \\ 9 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ 4.5 \\ 3 \\ 2 \end{pmatrix}$$

Scalar Multiplication

Multiplying a matrix by a scalar (a single number) involves multiplying each element of the matrix by that scalar.

Example: Continuing from the user preference vector $\mathbf{u}_1$:

$$2\mathbf{u}_1 = 2\begin{pmatrix} 5 \\ 4 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 \times 5 \\ 2 \times 4 \\ 2 \times 2 \\ 2 \times 3 \end{pmatrix} = \begin{pmatrix} 10 \\ 8 \\ 4 \\ 6 \end{pmatrix}$$

Scalar multiplication can be used to scale features or adjust the importance of certain components.

Dot Product (Inner Product) and Cosine Similarity

The *dot product* of two vectors $\mathbf{u}$ and $\mathbf{v}$ of the same dimension is a scalar value, calculated as:

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \sum_i u_i v_i$$

While the dot product itself provides a measure of similarity, it is sensitive to the magnitude (length) of the vectors. To focus on the *direction* or *pattern* of the vectors, irrespective of their magnitude, we often use *cosine similarity*, which incorporates normalization.

Cosine similarity is calculated by normalizing the vectors and then taking their dot product. The normalization step involves dividing each vector by its Euclidean norm (magnitude):

$$\text{Cosine Similarity}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||}$$

where $||\mathbf{u}|| = \sqrt{\sum_i u_i^2}$ is the Euclidean norm of vector $\mathbf{u}$, and similarly for $||\mathbf{v}||$.

Example (cosine similarity): Let's calculate the cosine similarity between user preference vectors $\mathbf{u}_1$ and $\mathbf{u}_2$:

$$\mathbf{u}_1 = \begin{pmatrix} 5 \\ 4 \\ 2 \\ 3 \end{pmatrix}, \quad \mathbf{u}_2 = \begin{pmatrix} 3 \\ 5 \\ 4 \\ 1 \end{pmatrix}$$

First, we calculate the dot product:

$$\mathbf{u}_1 \cdot \mathbf{u}_2 = (5 \times 3) + (4 \times 5) + (2 \times 4) + (3 \times 1) = 15 + 20 + 8 + 3 = 46$$

Next, we calculate the Euclidean norm (magnitude) of each vector:

$$||\mathbf{u}_1|| = \sqrt{5^2 + 4^2 + 2^2 + 3^2} = \sqrt{25 + 16 + 4 + 9} = \sqrt{54} \approx 7.35$$

$$||\mathbf{u}_2|| = \sqrt{3^2 + 5^2 + 4^2 + 1^2} = \sqrt{9 + 25 + 16 + 1} = \sqrt{51} \approx 7.14$$

Finally, we compute the cosine similarity:

$$\text{Cosine Similarity}(\mathbf{u}_1, \mathbf{u}_2) = \frac{46}{7.35 \times 7.14} \approx \frac{46}{52.5} \approx 0.876$$

The cosine similarity between $\mathbf{u}_1$ and $\mathbf{u}_2$ is approximately 0.876. Cosine similarity values range from -1 to 1, where 1 indicates perfect similarity (same direction), 0 indicates orthogonality (no similarity in direction), and -1 indicates opposite directions. A higher cosine similarity indicates a greater degree of similarity in the pattern of preferences, *normalized for the scale of ratings*. Using cosine similarity is often more appropriate than just the raw dot product when we want to compare the direction of preferences irrespective of how strongly users tend to rate items in general.

Matrix Multiplication

The product of two matrices $\mathbf{A}$ (of size $m \times p$) and $\mathbf{B}$ (of size $p \times n$) is a matrix $\mathbf{C} = \mathbf{AB}$ (of size $m \times n$). The element C_{ij} is calculated as the *dot product* of the i-th row of $\mathbf{A}$ and the j-th column of $\mathbf{B}$:

$$C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}$$

Example (simplified content-based prediction): Suppose we have a user feature matrix $\mathbf{U}$ (users vs. genre preferences) and an item feature matrix $\mathbf{I}$ (genres vs. movies, representing how much each movie belongs to each genre). Let's say (simplified for illustration):

$$\mathbf{U} = \left(\begin{array}{cccc} & \text{Action} & \text{Comedy} & \text{Drama} \\ \text{User 1} & 5 & 4 & 2 \\ \text{User 2} & 3 & 5 & 4 \end{array} \right)$$

$$\mathbf{I} = \left(\begin{array}{ccc} & \text{Movie X} & \text{Movie Y} \\ \text{Action} & 0.7 & 0.2 \\ \text{Comedy} & 0.1 & 0.6 \\ \text{Drama} & 0.2 & 0.2 \end{array} \right)$$

Then, the predicted user-item preference matrix $\mathbf{P} = \mathbf{UI}$ can be (approximately) calculated using matrix multiplication. For example, the predicted preference of User 1 for Movie X, P_{11}, is:

$$P_{11} = (5 \times 0.7) + (4 \times 0.1) + (2 \times 0.2) = 3.5 + 0.4 + 0.4 = 4.3$$

Note: This is a highly simplified illustration of content-based prediction using matrix multiplication.

2.1.3 Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors are important concepts in linear algebra, particularly relevant to Singular Value Decomposition (SVD), a technique sometimes used in recommender systems. While modern recommender systems often use more scalable methods, understanding eigenvalues and eigenvectors provides a foundation for grasping dimensionality reduction and latent factor models, which are conceptually related to Matrix Factorization.

Eigenvector

An eigenvector $\mathbf{v}$ of a square matrix $\mathbf{A}$ is a non-zero vector that, when multiplied by $\mathbf{A}$, results in a vector that is a scalar multiple of $\mathbf{v}$. It represents a direction in space that is only scaled, not rotated, by the linear transformation represented by $\mathbf{A}$.

Eigenvalue

The scalar multiple is called the eigenvalue λ, associated with the eigenvector $\mathbf{v}$. The relationship is defined by:

$$\mathbf{Av} = \lambda \mathbf{v}$$

Eigenvalues indicate the magnitude of scaling in the direction of their corresponding eigenvectors.

Example

Imagine we have a matrix $\mathbf{M}$ that represents movie genre relationships. In this example, let's consider $\mathbf{M}$ to be a *Genre-Genre Relationship Matrix*. Both the rows and columns of $\mathbf{M}$ represent the set of movie genres we are considering (e.g., Action, Comedy, Drama, Sci-Fi, Romance, etc.). So, if we have n genres, $\mathbf{M}$ will be an $n \times n$ matrix. Let's simplify for this example and consider just 4 genres:

- Genre 1: Action (A)
- Genre 2: Comedy (C)
- Genre 3: Sci-Fi (SF)
- Genre 4: Drama (D)

Then $\mathbf{M}$ will be a 4×4 matrix and the entry M_{ij} in the matrix $\mathbf{M}$ (at row i, column j) represents the strength of the relationship or association between genre i and genre j. Let's consider that M_{ij} in this numerical example represents the *(normalized) average co-occurrence count* of genre i and genre j in movie descriptions across a large dataset (higher value = more frequent co-occurrence). A possible numerical example for $\mathbf{M}$ could be:

$$\mathbf{M} = \begin{pmatrix} & \text{A} & \text{C} & \text{SF} & \text{D} \\ \text{A} & 1.0 & 0.2 & 0.9 & 0.1 \\ \text{C} & 0.2 & 1.0 & 0.1 & 0.4 \\ \text{SF} & 0.9 & 0.1 & 1.0 & 0.2 \\ \text{D} & 0.1 & 0.4 & 0.2 & 1.0 \end{pmatrix}$$

Interpretation of Numerical Values in $\mathbf{M}$:

- $M_{11} = 1.0$: Relationship of Action with Action itself is set to 1.0 (for normalization, diagonal usually has higher values).
- $M_{13} = M_{31} = 0.9$: Relatively high value between Action (Genre 1) and Sci-Fi (Genre 3), indicating frequent co-occurrence. Many movies are tagged as both Action and Sci-Fi.
- $M_{24} = M_{42} = 0.4$: Moderate value between Comedy (Genre 2) and Drama (Genre 4), indicating some co-occurrence (e.g., "dramedy").
- $M_{12} = M_{21} = 0.2$: Low value between Action (Genre 1) and Comedy (Genre 2), suggesting they don't co-occur as often. And similarly low value between Sci-Fi (Genre 3) and Drama (Genre 4), $M_{34} = M_{43} = 0.2$; and between Action (Genre 1) and Drama (Genre 4), $M_{14} = M_{41} = 0.1$.
- Diagonal elements ($M_{ii} = 1.0$): Set to 1.0 for each genre's relationship with itself (can be seen as maximum self-similarity or for normalization).
- Matrix is symmetric ($M_{ij} = M_{ji}$): Relationship between genre i and j is assumed to be symmetric.

Eigenvectors of $\mathbf{M}$ might then represent dominant genre combinations or latent genre factors that capture underlying patterns in movie preferences and genre relationships. For instance, if we perform eigenvalue decomposition on $\mathbf{M}$, we might find that the largest eigenvalue is 2.04 and corresponding eigenvector is approximately:

$$\mathbf{v}_1 \approx \begin{pmatrix} 0.64 \\ 0.30 \\ 0.64 \\ 0.30 \end{pmatrix}$$

This eigenvector $\mathbf{v}_1$ has relatively high values for the Action (first component, 0.64) and Sci-Fi (third component, 0.64) genres, and lower values for Comedy and Drama. This could be interpreted as representing a "major action-adventure theme" that combines elements of Action and Sci-Fi, with less emphasis on Comedy and Drama. The corresponding eigenvalue ($\lambda_1 = 2.04$, would indicate the "strength" or "importance" of this theme in the overall movie genre landscape captured by $\mathbf{M}$.

Movies that are strongly aligned with this eigenvector (i.e., have a high projection onto this eigenvector direction) would be considered strongly representative of this "action-adventure theme." For example, a movie with genre proportions somewhat similar to $\mathbf{v}_1$ (more Action and Sci-Fi, less Comedy and Drama) would have a higher projection onto $\mathbf{v}_1$.

Here, we do not delve into the manual calculation of eigenvalues and eigenvectors. Readers interested in the computational aspect can refer to Listing 2.1 for a Python code example demonstrating their computation. It is important to note that while direct computation of eigenvalues and eigenvectors is not typically a core component of most practical recommender system algorithms we will encounter later in this book, understanding these concepts provides a valuable theoretical foundation. This foundation is crucial for grasping dimensionality reduction and latent factor models, which *are* central to techniques like *Matrix Factorization* that we will discuss in subsequent chapters. Consider eigenvalues and eigenvectors, in this context, as providing an intuitive understanding of how we can uncover hidden structure and important dimensions within complex data – a core principle underlying many recommender system techniques.

```python
import numpy as np

# define the M matrix
m = np.array([
    [1.0, 0.2, 0.9, 0.1],
    [0.2, 1.0, 0.1, 0.4],
    [0.9, 0.1, 1.0, 0.2],
    [0.1, 0.4, 0.2, 1.0]
])

# compute the eigenvalue and eigenvector
val, vec = np.linalg.eig(m)
print('Eigenvalue:', val)
print('Eigenvector', vec)
```

Listing 2.1: Example Python Code to Compute Eigenvalue and Eigenvector

2.2 Probability and Statistics

Basic concepts from probability and statistics are essential for understanding evaluation metrics and some probabilistic models used in recommender systems.

2.2.1 Descriptive Statistics

Mean (Average)

Mean (or average) can be obtained by dividing the sum of values by the number of values. For a dataset of n values $\{x_1, x_2, \ldots, x_n\}$, the mean μ is calculated as:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

Example: Suppose we have collected movie ratings from a user: [4, 5, 3, 4, 5]. The *mean* rating is $\frac{4+5+3+4+5}{5} = \frac{21}{5} = 4.2$. The average rating gives a central tendency of the user's movie preferences.

Median

Median is the middle value in a sorted dataset. To find the median:

1. Sort the dataset in ascending order.
2. If the dataset has an odd number of values, the median is the middle value.
3. If the dataset has an even number of values, the median is typically the average of the two middle values.

Example: For the same ratings [4, 5, 3, 4, 5], first sort them: [3, 4, 4, 5, 5]. The *median* rating is *4* (the middle value). The median is less sensitive to extreme values than the mean.

Variance

Variance is a measure of the average squared deviation from the mean; it quantifies the spread or dispersion of data points around the mean.

Population Variance: For a population dataset of n values $\{x_1, x_2, \ldots, x_n\}$ with mean μ, the *population variance* σ^2 is:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

Sample Variance: For *sample variance*, the denominator is typically $n-1$ (instead of n) to provide an *unbiased* estimate of the population variance from a sample, but for conceptual understanding, the population variance formula is often introduced first, as it is more straightforward.[1]

Standard Deviation

Standard deviation is the *square root* of the variance. It is also a measure of the spread or dispersion of data around the mean, but is in the original units of the data, making it often more interpretable than variance.

Population Standard Deviation: The population standard deviation σ is the square root of the population variance σ^2:

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

Example: Comparing two users:

- User A's ratings: [3, 3, 4, 4, 4]. Mean = 3.6, Standard Deviation (approx.) = 0.49.
- User B's ratings: [1, 3, 4, 5, 5]. Mean = 3.6, Standard Deviation (approx.) = 1.50.

Both users have the same mean rating, but User B's ratings have *higher* standard deviation, indicating *more variability* in their ratings (more diverse tastes or less consistent rating behavior).

Percentiles and Quartiles

Percentiles and quartiles are values that divide a dataset into hundredths (*percentiles*) or quarters (*quartiles*); they are useful for understanding data distribution.

Percentile: The P^{th} percentile, often denoted as P_{th}, is the value below which approximately $P\%$ of the data falls.

Example: Consider all movie ratings in a system. The *25th percentile* rating might be 3, meaning approximately 25% of all ratings are below or equal to 3. The *75th percentile* rating might be 4.5, meaning approximately 75% of ratings are below or equal to 4.5. The *interquartile range* (IQR) is defined as the difference between the 75th percentile (Q_3) and the 25th percentile (Q_1): $IQR = Q_3 - Q_1$. It gives a measure of the *spread* of the middle 50% of the data.

2.2.2 Basic Probability

Probability

Probability is a measure of the likelihood of an event occurring, ranging from 0 (impossible) to 1 (certain). For example, consider user clicks on recommended items. If, on average, 20 out of 100 recommended items are clicked by a user, we could say the *probability of a click* for a randomly chosen recommendation is approximately 0.2 or 20%.

Random Variables

Random Variables are variables whose values are numerical outcomes of a random phenomenon. For example, let X be a random variable representing the *rating a user gives to a movie* on a scale of 1 to 5. The value of X is *random* until the user actually provides a rating. Possible values for X are {1, 2, 3, 4, 5}.

[1] For more detailed discussion on unbiased estimate of variance, see Appendix Section **??**.

Probability Distribution

Probability Distributions describe the likelihood of different outcomes for a random variable. Here are some typical examples:

- **Bernoulli Distribution** is probability distribution of a random variable that can take one of two values, typically 0 or 1 (e.g., success/failure, click/no-click).

  Probability Mass Function (PMF) gives the probability of a *discrete* random variable taking on a specific value. For a Bernoulli random variable Y with probability of success p, the PMF is:

  $$P(Y=y)=\begin{cases} p & \text{if } y=1 \text{ (success)} \\ 1-p & \text{if } y=0 \text{ (failure)} \\ 0 & \text{otherwise} \end{cases}$$

  Here, y represents the possible outcomes of the Bernoulli random variable Y, which are typically 0 (failure) or 1 (success).

  Example: Let Y be a random variable representing whether a user *clicks (1)* or *does not click (0)* on a recommended ad. If the probability of a click is 0.3, then Y follows a Bernoulli distribution with parameter $p=0.3$. For example, $P(Y=1)=0.3$ and $P(Y=0)=0.7$.

- **Binomial Distribution** is probability distribution of the number of successes in a fixed number of independent Bernoulli trials. For a Binomial random variable Z representing the number of successes in n trials, with probability of success p in each trial, the *PMF* is:

  $$P(Z=k)=\binom{n}{k}p^k(1-p)^{n-k}$$

  where $\binom{n}{k}=\frac{n!}{k!(n-k)!}$ is the binomial coefficient, and k represents *the number of successes* (out of n trials).

  Example: Consider a recommender system displaying 10 recommendations. Let Z be a random variable representing the *number of clicks out of these 10 recommendations*. If each recommendation has a probability of click of 0.2 (and clicks are independent), then Z follows a Binomial distribution with parameters $n=10$ and $p=0.2$. For example, $P(Z=3)$ would be the probability of getting exactly 3 clicks out of 10 recommendations, calculated using the formula with $k=3, n=10, p=0.2$.

- **Normal (Gaussian) Distribution:** A continuous probability distribution characterized by its bell-shaped curve, often used to model real-valued data.

  Probability Density Function (PDF) is used for *continuous* random variables. For a Normal random variable Z with mean μ and standard deviation σ, the PDF is:

  $$f(z)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2}$$

  Here, z is a value of the continuous random variable Z. And $\pi\approx 3.14159$ and $e\approx 2.71828$ are mathematical constants. Please note that for a continuous distribution, the value of the PDF at a specific point *does not* directly give a probability. Instead, probabilities are represented by the *area under the PDF curve* over a range of values (more discussion on this in Appendix Section **??**). Detailed understanding of this formula itself is not crucial at this introductory stage, focus on the concept of Normal distribution and its bell shape (see also Figure 2.1).

  Example: While ratings are discrete, sometimes, when dealing with predicted ratings or errors in prediction, we might assume they are approximately normally distributed. For instance, the *error in predicting a user's rating* for a movie might be modeled as a random variable following a normal distribution centered around zero (i.e., $\mu=0$).

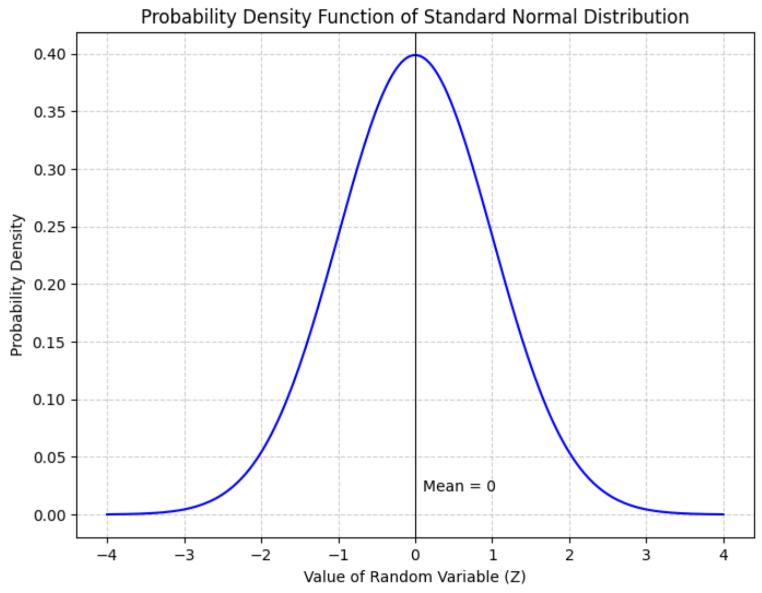

Figure 2.1: Probability density function of the Normal distribution with $\mu = 0$ and $\sigma = 1$, illustrating its characteristic bell shape.

Conditional Probability

Conditional Probability is the probability of an event A occurring *given that* another event B has already occurred. The conditional probability of event A given event B, denoted as $P(A|B)$, is defined as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

where $P(A \cap B)$ is the probability of both A and B occurring, and $P(B) > 0$.

Bayes' Theorem

Bayes' Theorem is a fundamental theorem that describes how to update our belief or probability about an event based on new evidence. In Bayesian statistics, we often talk about *prior* probability and *posterior* probability.

Prior probability $(P(A))$ is our *initial belief* or probability about an event A *before* observing any new evidence. It represents our baseline knowledge or assumption.

Posterior probability $(P(A|B))$ is the *updated probability* of event A *after* considering new evidence B. It represents our revised belief in light of the evidence. Bayes' Theorem provides a way to calculate the posterior probability from the prior probability and the evidence.

Bayes' Theorem formula:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Example: Let's reconsider the movie genre example to illustrate *prior* and *posterior* probabilities in the context of Bayes' Theorem. Suppose:

- Event A: User likes Action movies.
- Event B: User likes Sci-Fi movies.

Assume we have the following probabilities:

- $P(A) = 0.2$ (*prior* probability of liking Action movies, i.e., *before* knowing anything about a specific user's movie preferences, we estimate that 20% of users in general like Action movies. This is our *prior* belief).
- $P(B) = 0.3$ (*prior* probability of liking Sci-Fi, i.e., similarly, 30% of users in general like Sci-Fi movies. This is also a *prior* belief).
- $P(B|A) = 0.6$ (*conditional* probability, i.e., of users who like Action movies (A), 60% also like Sci-Fi movies (B). This is the *likelihood* of observing evidence *B* given *A* is true).

We want to calculate $P(A|B)$: "What is the *posterior* probability that a user likes Action movies (A) *given that* we now have evidence that they like Sci-Fi movies (B)?"

Using Bayes' Theorem:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} = \frac{0.6 \times 0.2}{0.3} = \frac{0.12}{0.3} = 0.4$$

So, the *posterior* probability $P(A|B) = 0.4$ or 40%.

Interpretation: Initially, our prior probability of a user liking Action movies was 20% ($P(A) = 0.2$). However, after observing the evidence that this user likes Sci-Fi movies (event B), we used Bayes' Theorem to *update* our probability to 40% ($P(A|B) = 0.4$). The *posterior* probability (40%) is higher than the *prior* probability (20%), reflecting the increased likelihood of the user liking Action movies given the new evidence. Bayes' Theorem provides a formal way to revise our beliefs in the face of new data.

We may touch upon this concept indirectly in some advanced techniques later, particularly in *Bayesian approaches* to recommender systems, where prior beliefs can be combined with observed data to refine recommendations. A brief overview about it is available in Section **??**, towards the end of this chapter.

2.2.3 Hypothesis Testing

Hypothesis testing is a framework for making decisions based on data. In recommender system evaluation, especially in *online A/B testing*, hypothesis testing is used to determine if *observed differences* in performance between different recommendation algorithms are *statistically significant* or just due to random chance.

Null Hypothesis and Alternative Hypothesis

Suppose we want to compare two recommender algorithms, Algorithm A (current system) and Algorithm B (new algorithm), based on their click-through rate (CTR). We want to test if Algorithm B actually improves CTR compared to Algorithm A.

- **Null Hypothesis ($\mathbf{H_0}$)** would state that there is *no difference* in the average CTR between Algorithm A and Algorithm B. In other words, any observed difference is due to random chance. Formally, we might state $H_0 : \mu_A = \mu_B$, where μ_A and μ_B are the true average CTRs of Algorithm A and Algorithm B, respectively.
- **Alternative Hypothesis ($\mathbf{H_1}$)** would state that Algorithm B has a *higher* average CTR than Algorithm A. We are trying to find evidence to support this claim. Formally, $H_1 : \mu_B > \mu_A$, i.e., a one-tailed test, as we are specifically interested in improvement. We could also have a two-tailed alternative, $H_1 : \mu_A \neq \mu_B$, if we were just testing for *any* difference.

P-Value

The probability of observing data as extreme as, or more extreme than, the observed data if *the null hypothesis were true*.

Example: Continuing the example above, let us say that after running an A/B test with Algorithm A and Algorithm B on a sample of users for a period, we measure the CTR for both. Suppose we observe:

- Algorithm A: CTR = 0.25 (25%)
- Algorithm B: CTR = 0.27 (27%)

We then perform a *statistical test* (e.g., a t-test or z-test, depending on assumptions) to calculate the p-value. Let us say here, just for an example, the test results in a p-value of 0.03.[2] This p-value of 0.03 means that if there were truly no difference in CTR between Algorithm A and Algorithm B (i.e., the Null Hypothesis is true), then there is only a 3% probability of observing a CTR difference as large as (or larger than) the 2% difference we observed in our experiment (0.27 - 0.25 = 0.02).

Statistical Significance

When the p-value is *below* a chosen significance level (often denoted as α, commonly set to 0.05), we reject the null hypothesis and conclude that the observed effect is *statistically significant*.

Illustrative example on decision making based on p-value:

- We set a significance level $\alpha = 0.05$ (common choice).
- We calculate the p-value, and it is 0.03. Since 0.03 is *less than* 0.05 (p-value $< \alpha$):
  - we *reject* the Null Hypothesis, and
  - we conclude that we have *statistically significant* evidence to suggest that Algorithm B has a *higher* CTR than Algorithm A at the 0.05 significance level, and
  - this supports adopting Algorithm B as the new recommendation system.
- *What if* the p-value was *higher*, e.g., 0.15?
  - If the p-value were 0.15, which is higher than 0.05 (p-value $> \alpha$), we would *fail to reject* the Null Hypothesis. It means:
    - we do not have enough statistical evidence to conclude that Algorithm B is significantly better than Algorithm A in terms of CTR, and
    - the observed 2% difference in CTR could reasonably be due to random chance and
    - we would not confidently switch to Algorithm B based on this experiment alone, and
    - further investigation or larger experiments might be needed.

2.3 Machine Learning Basics

Many techniques used in Recommender Systems can be categorized into machine learning. A basic understanding of machine learning concepts will provide context for the algorithms we will discuss. Machine learning paradigms can be broadly categorized based on the type of data and learning signals used.

2.3.1 Supervised Learning

Supervised learning involves learning a mapping function f from input features $\mathbf{x}$ to output labels y (or target values) based on labeled training data $(\mathbf{x}_i, y_i)$. The goal is to learn a model that can predict y for new, unseen input $\mathbf{x}$. Supervised learning relies on explicitly labeled data where both inputs and desired outputs are provided during training.

[2] For more detailed discussion on p-value, see Appendix Section **??**.

Classification

Classification aims to predict a categorical label $y \in \{c_1, c_2, ..., c_k\}$ from input features $\mathbf{x}$. The output y belongs to a discrete set of classes.

Definition: Learn a function $f(\mathbf{x})$ such that $f(\mathbf{x})$ predicts the class label y for a given input $\mathbf{x}$. The model outputs a class from a predefined set of categories.

Example: Given movie features (e.g., keywords in plot synopsis, actors, director), build a classifier to predict the *genre* of a movie (e.g., Action, Comedy, Drama).

- *Input features $\mathbf{x}$ for a movie:* Could be an embedding representation of keywords from the plot synopsis, or binary features indicating presence of certain actors.
- *Output label y:* One of the genres: "Action", "Comedy", "Drama", "Sci-Fi", etc.
- *Training data:* A dataset of movies where each movie is labeled with its genre(s) and has associated features.
- *Goal:* Train a classification model (e.g., Logistic Regression, Support Vector Machine, Decision Tree, (Deep) Neural Network) to predict the genre of a *new, unseen movie* based on its features.

Regression

Regression aims to predict a continuous numerical value $y \in \mathbb{R}$ from input features $\mathbf{x}$. The output y is a real number.

Definition: Learn a function $g(\mathbf{x})$ such that $g(\mathbf{x})$ predicts a real-valued output y for a given input $\mathbf{x}$.

Example: Given user features (e.g., demographics, past rating history) and movie features (e.g., genre, actors, content features), build a regression model to predict the **rating** a user would give to a movie (e.g., on a scale of 1 to 5).

- *Input features $\mathbf{x}$ for a user-movie pair:* Could be a combination of user features and movie features. For example, user's age, user's genre preferences, movie's genre, movie's average rating by other users.
- *Output value y:* Predicted rating, a real number (e.g., 4.5, 3.2, etc., often rounded to the rating scale if needed).
- *Training data:* A dataset of user-movie pairs where we know the actual rating a user gave to a movie and we have the associated features.
- *Goal:* Train a regression model (e.g., Linear Regression, Regression Tree, (Deep) Neural Network) to predict the rating a user would give to a *movie they have not rated yet*, based on their features and the movie's features. This is directly related to recommender systems that predict ratings.

2.3.2 Unsupervised Learning

Unsupervised learning deals with finding patterns and structure in unlabeled data $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, without explicit output labels. The learning process is driven by the inherent structure of the data itself, aiming to discover hidden patterns and relationships.

Clustering

Clustering aims to group similar data points together into clusters. The goal is to partition the data into meaningful subgroups based on similarity.

Definition: Given a set of data points $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, partition them into k clusters $C = \{C_1, C_2, ..., C_k\}$ such that data points within the same cluster are more similar to each other than to points in other clusters, according to a defined similarity or distance measure.

Example: Cluster users based on their movie viewing history and ratings to identify *user segments with similar tastes*.

- *Input Data* $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$*:* Each $\mathbf{x}_i$ could be a vector representing user i's movie ratings (e.g., a row from the user-item rating matrix, or user genre preference vector).

- *Clustering Algorithm:* Apply a clustering algorithm like K-Means, DBSCAN, or Hierarchical Clustering to group users.

- *Output Clusters* $C = \{C_1, C_2, ..., C_k\}$*:* Groups of users who have exhibited similar movie preferences. For example, Cluster 1 might be "Action Movie Fans," Cluster 2 "Comedy Lovers," etc.

- *Use in Recommender Systems:* Once user clusters are identified, you can tailor recommendations to each cluster. For instance, recommend action movies to users in the "Action Movie Fans" cluster.

Dimensionality Reduction

Dimensionality reduction aims to reduce the number of variables (dimensions) in data while preserving essential information. The goal is to represent high-dimensional data in a lower-dimensional space, often to simplify computation, visualization, or feature extraction.

Definition: Given high-dimensional data $X \in \mathbb{R}^{N \times D}$ (N data points, D dimensions), find a transformation to map it to a lower-dimensional space $Y \in \mathbb{R}^{N \times d}$ where $d < D$, while retaining as much variance or information as possible.

Example: Suppose movies are initially described by a very large number of features (e.g., embeddings of plot words, actor embeddings, visual features). Use dimensionality reduction techniques like Principal Component Analysis (PCA) to *reduce the number of movie features* while retaining the most important information for similarity comparisons and recommendation.

- *Input Data* X*:* A matrix where rows are movies and columns are original high-dimensional movie features.

- *Dimensionality Reduction Algorithm:* Apply PCA or similar method.

- *Output* Y*:* A new matrix where rows are still movies, but columns are now a smaller set of *lower-dimensional features* (principal components). These lower-dimensional features can capture the most important underlying characteristics of movies in a compressed form and can be used for more efficient content-based recommendation algorithms or as input to other models.

2.3.3 Semi-Supervised Learning

Semi-supervised learning bridges the gap between supervised and unsupervised learning. It leverages both *labeled data* and a larger amount of *unlabeled data* during training. The core idea is that even without explicit labels for all data points, the underlying structure in the unlabeled data can guide and improve the learning process, especially when labeled data is limited or costly to acquire. In the context of recommender systems, we often encounter scenarios where *explicit ratings (labeled data)* are sparse, but *implicit feedback (unlabeled data)*, such as user browsing history, clicks, views, or dwell time, is abundant and easier to collect.

Definition

Let $D_{labeled} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{L}$ represent a set of L *labeled data points*, and $D_{unlabeled} = \{\mathbf{x}'_j\}_{j=1}^{Unl}$ represent a set of Unl *unlabeled data points*.

Objective

In semi-supervised learning, we aim to learn a model f that leverages both $D_{labeled}$ and $D_{unlabeled}$. This is typically achieved by minimizing a loss function that includes a *supervised loss component* (measured on $D_{labeled}$ to ensure the model learns to predict the labels correctly where available) and an *unsupervised component* (measured on $D_{unlabeled}$ to encourage the model to learn meaningful representations from the unlabeled data distribution). The unsupervised component can take various forms, such as regularization terms, consistency constraints, or pseudo-labeling techniques. The overall goal is to improve performance on the supervised prediction task by exploiting the information present in the unlabeled data.

Use Cases in Recommender Systems

Recommender systems often have sparse explicit ratings but abundant implicit feedback. Semi-supervised learning provides techniques to effectively utilize this readily available implicit feedback to enhance recommendation models, e.g., for improving rating prediction accuracy or item ranking.

The idea is to train models using a combination of a limited amount of explicitly labeled data (e.g., user ratings) and a larger quantity of unlabeled data (e.g., user browsing history, item views). The motivation to do so is that the unlabeled data might help the model to learn the underlying structure and distribution of the input data. This learned structure can then improve the model's ability to generalize and make more accurate predictions, even for the primary supervised task (like rating prediction).

Concrete Example

Consider a movie recommender system where the primary task is to predict user ratings for movies (supervised task). We have a dataset of explicit movie ratings (e.g., 1-5 stars), which is our *labeled data* $D_{labeled}$. However, we also have a much larger dataset of user movie *viewing history* (which movies users have watched, but without explicit ratings), which serves as our *unlabeled data* $D_{unlabeled}$. We can use semi-supervised learning to enhance our rating prediction model by leveraging this unlabeled viewing data.

- *Labeled data* $D_{labeled}$*:* User-movie pairs with explicit ratings (e.g., (User A, Movie X, 4 stars), (User B, Movie Y, 2 stars)). These are used for the supervised loss component to learn to predict ratings.
- *Unlabeled data* $D_{unlabeled}$*:* User-movie pairs indicating user viewed a movie, but no explicit rating given (e.g., (User C, Movie Z), (User D, Movie W)). This viewing data, while not providing explicit ratings, implicitly signals user interest and preference. It's used in the unsupervised component of the loss function.
- *Semi-supervised learning:* Train a model to predict ratings using the labeled data (supervised loss), and employ that same model to learn from the viewing patterns in the unlabeled data. For example, we might encourage the model to learn user and movie representations such that users who have viewed similar sets of movies (in the unlabeled data) are also predicted to have similar rating preferences (in the labeled data). Techniques like enforcing *smoothness assumptions* (i.e., users with similar viewing patterns should have similar rating functions) or *pseudo-labeling* (i.e., generating "pseudo-labels" from the model's predictions on unlabeled data and using them for further training) could be employed to leverage the unlabeled data.
- *Goal:* Build a rating prediction model that achieves improved accuracy and generalization by effectively combining information from both the limited explicit ratings (labeled data) and the more abundant implicit viewing data (unlabeled data). The focus remains on improving the supervised task of rating prediction.

2.3.4 Self-Supervised Learning

Coming soon...

Part II

Core Recommender System Techniques

Chapter 3

Content-Based Recommender Systems

Coming soon...