Dr.BGR Publications

Recommender Systems

Nancy Chitra Thilaga N

Recommender Systems

Nancy Chitra Thilaga N

Assistant Professor

Grace College of Engineering

Thoothukudi

Dr.BGR
Publications

Verso Page

Publishing House	Dr. BGR Publications
	Tuticorin - 05
	9003494749
	drbgrpublications@gmail.com
	https://drbgrpublications.in/books/
	https://www.instagram.com/drbgrpublications/
Title	Recommender Systems
ISBN	978-81-989720-3-3
Language	English
Country of Publication	India
Product Composition	Single-Component Retail Product
Product Form	Digital download and online
First published	2025
Date of Publication	09/08/2025
Access Type	Open Access (OA)
Author	Nancy Chitra Thilaga N
Copyright	© Author
Edited and typeset by	Dr. BGR Publications
Cover design by	Dr. B.Govindarajan
Bar Code	

For permission requests, write to the author at nancychitrathilaga@grace.edu.in

Disclaimer

The authors are solely responsible for the contents of the book. The publishers or editors do not take any responsibility for the same in any manner. Errors, if any, are purely unintentional and readers are requested to communicate such errors to the editors or publishers to avoid discrepancies in future.

About the Book

"RECOMMENDER SYSTEMS" is a comprehensive guide tailored for students, educators, and professionals interested in understanding the foundation and evolution of personalized recommendation technologies. Developed in alignment with the CCS360 curriculum, this book explores both the theoretical underpinnings and practical implementations of recommender systems in today's data-driven world.

Covering a range of topics including content-based filtering, collaborative filtering, hybrid methods, deep learning approaches, and real-world applications, the book balances clarity with technical depth, making it suitable for both beginners and advanced learners. With illustrative examples, algorithms, and exercises, this book serves as a reliable resource for mastering one of the most impactful fields in AI and machine learning.

Whether you're a university student preparing for exams, a teacher seeking structured material, or a data enthusiast eager to explore personalization algorithms, this book is your go-to guide for building and understanding modern recommender systems.

About the Author

N. Nancy Chitra Thilaga, M.E. in Applied Electronics, is an Assistant Professor at Grace College of Engineering, Thoothukudi. She brings a unique blend of academic insight and industry experience, having spent four years in the IT sector before transitioning to academia. Her exposure to real-world technological challenges enhances both her teaching and writing.

With a strong academic foundation and a deep interest in emerging technologies, she delivers content that is both conceptually clear and practically relevant. Her focus on curriculum-aligned material and application-oriented perspectives makes this book a valuable contribution to the academic literature in the fields of recommender systems and artificial intelligence.

Dr.BGR Publications

Index

Dr.BGR Publications

Syllabus

UNIT 1 INTRODUCTION

Introduction and basic taxonomy of recommender systems - Traditional and non-personalized Recommender Systems - Overview of data mining methods for recommender systems- similarity measures- Dimensionality reduction Singular Value Decomposition (SVD)

UNIT II CONTENT-BASED RECOMMENDATION SYSTEMS

High-level architecture of content-based systems - Item profiles, Representing item profiles, Methods for learning user profiles, Similarity-based retrieval, and Classification algorithms

UNIT III COLLABORATIVE FILTERING

A systematic approach, Nearest-neighbor collaborative filtering (CF), user-based and item-based CF, components of neighborhood methods (rating normalization, similarity weight computation, and neighborhood selection

UNIT IV ATTACK-RESISTANT RECOMMENDER SYSTEMS

Introduction - Types of Attacks - Detecting attacks on recommender systems - Individual attack - Group attack - Strategies for robust recommender design - Robust recommendation algorithms

UNIT V EVALUATING RECOMMENDER SYSTEMS

Evaluating Paradigms - User Studies - Online and Offline evaluation - Goals of evaluation design - Design Issues - Accuracy metrics - Limitations of Evaluation measures

ISBN: 978-81-989720-3-3

UNIT 1

INTRODUCTION

Introduction and basic taxonomy of recommender systems - Traditional and non-personalized Recommender Systems - Overview of data mining methods for recommender systems - similarity measures- Dimensionality reduction Singular Value Decomposition (SVD)

INTRODUCTION

Recommender systems aim to predict users' interests and recommend product items that quite likely are interesting for them. They are among the most powerful machine learning systems that online retailers implement in order to drive sales.

Data required for recommender systems stems from explicit user ratings after watching a movie or listening to a song, from implicit search engine queries and purchase histories, or from other knowledge about the users/items themselves.

Sites like Spotify, YouTube or Netflix use that data in order to suggest playlists, so-called Daily mixes, or to make video recommendations, respectively.

Need for Recommender Systems

Companies using recommender systems focus on increasing sales as a result of very personalized offers and an enhanced customer experience.

Recommendations typically speed up searches and make it easier for users to access content they're interested in, and surprise them with offers they would have never searched for.

What is more, companies are able to gain and retain customers by sending out emails with links to new offers that meet the recipients' interests, or suggestions of films and TV shows that suit their profiles.

The user starts to feel known and understood and is more likely to buy additional products or consume more content. By knowing what a user wants, the company gains competitive advantage and the threat of losing a customer to a competitor decreases.

Providing that added value to users by including recommendations in systems and products is appealing. Furthermore, it allows companies to position ahead of their competitors and eventually increase their earnings.

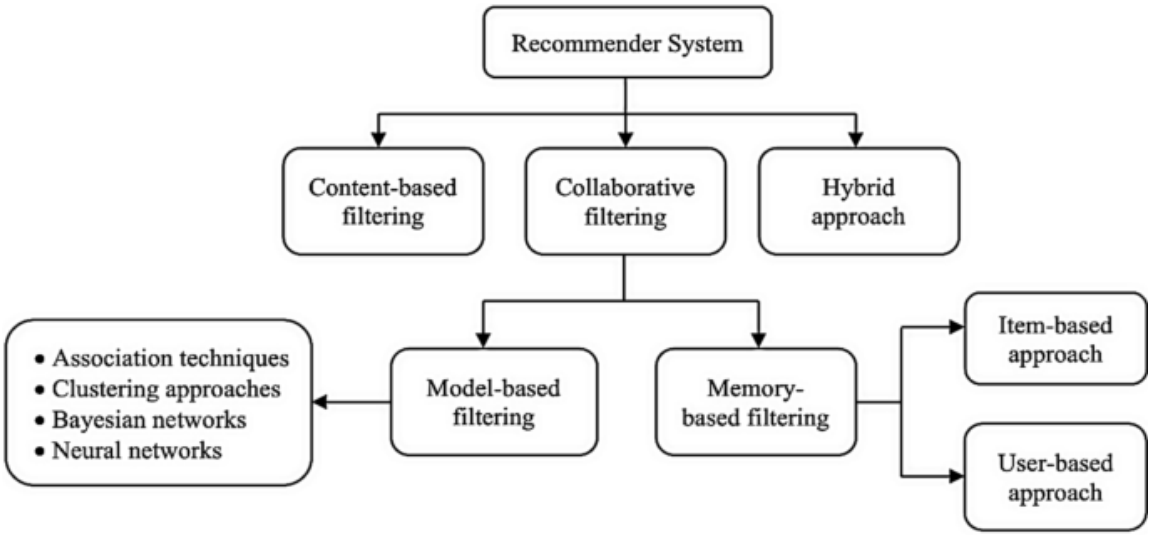
Recommender systems function with two kinds of information:

- *Characteristic information*. This is information about items (keywords, categories, etc.) and users (preferences, profiles, etc.).
- *User-item interactions.* This is information such as ratings, number of purchases, likes, etc.

ISBN: 978-81-989720-3-3

BASIC TAXONOMY OF RECOMMENDER SYSTEMS

- *Content-based* systems, which use characteristic information.
- *Collaborative filtering* systems, which are based on user-item interactions.
- *Hybrid systems*, which combine both types of information with the aim of avoiding problems that are generated when working with just one kind.



1. Content-based systems

These systems make recommendations using a user's item and profile features. They hypothesize that if a user was interested in an item in the past, they will once again be interested in it in the future. Similar items are usually grouped based on their features. User profiles are constructed using historical interactions or by explicitly asking users about their interests. There are other systems, not considered purely content-based, which utilize user personal and social data.

One issue that arises is making obvious recommendations because of excessive specialization (user A is only interested in categories B, C, and D, and the system is not able to recommend items outside those categories, even though they could be interesting to them).

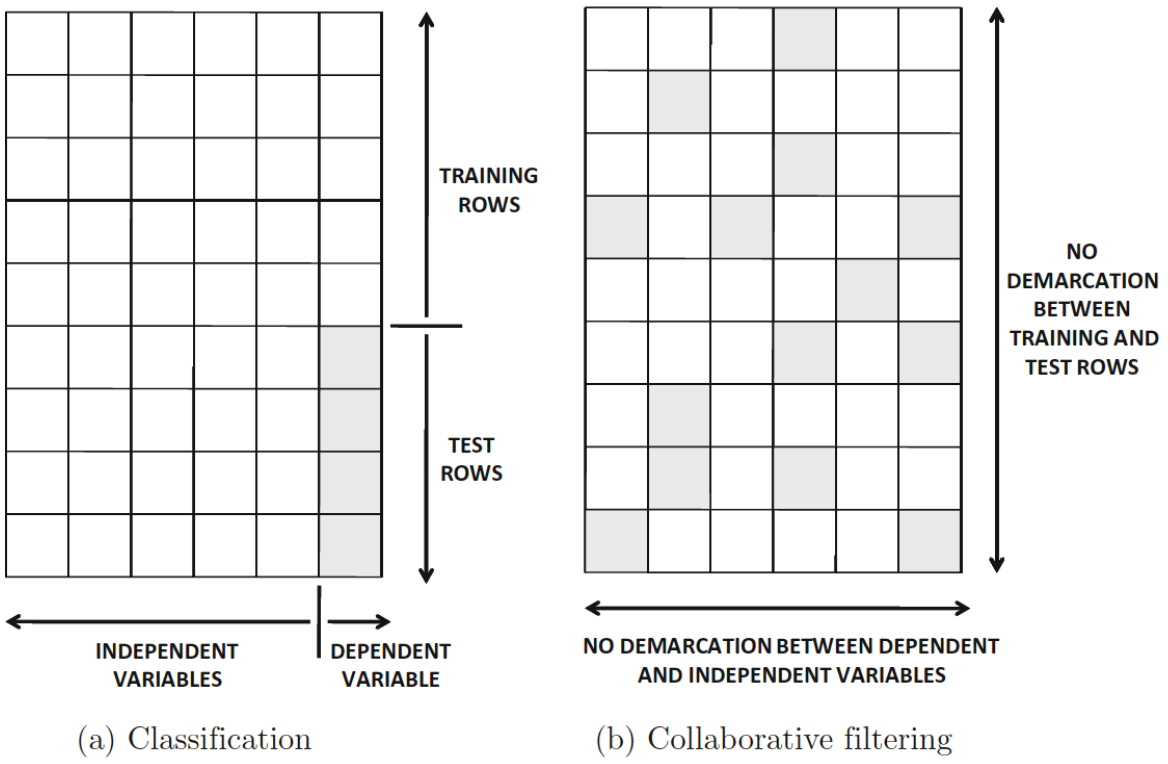
Another common problem is that new users lack a defined profile unless they are explicitly asked for information. Nevertheless, it is relatively simple to add new items to the system. We just need to ensure that we assign them a group according to their features.

2. Collaborative filtering systems

Collaborative filtering is currently one of the most frequently used approaches and usually provides better results than content-based recommendations. Some examples of this are found in the recommendation systems of Youtube, Netflix, and Spotify.

These kinds of systems utilize user interactions to filter for items of interest. We can visualize the set of interactions with a matrix, where each entry (i,j)(*i*,*j*) represents the interaction between user i*i* and item j*j*. An interesting way of looking at collaborative filtering

is to think of it as a generalization of classification and regression. While in these cases we aim to predict a variable that directly depends on other variables (features), in collaborative filtering there is no such distinction of feature variables and class variables. Visualizing the problem as a matrix, we don't look to predict the values of a unique column, but rather to predict the value of any given entry.



Classification vs Collaborative filtering

In short, collaborative filtering systems are based on the assumption that if a user likes item A and another user likes the same item A as well as another item, item B, the first user could also be interested in the second item. Hence, they aim to predict new interactions based on historical ones.

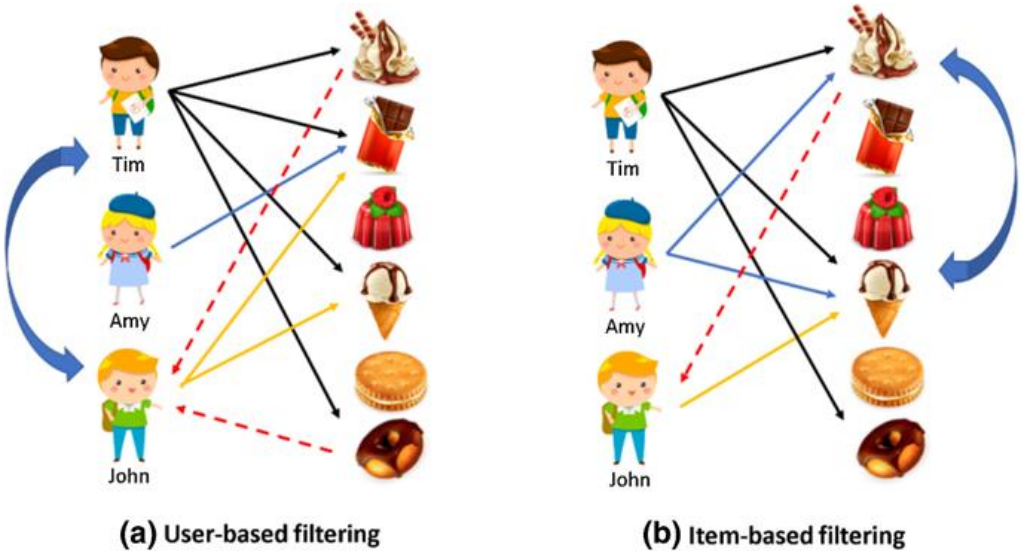
There are two types of methods to achieve this goal: *memory-based* and *model-based*.

Memory-based

There are two approaches: the first one identifies clusters of users and utilizes the interactions of one specific user to predict the interactions of other similar users. The second approach identifies clusters of items that have been rated by user A and utilizes them to predict the interaction of user A with a different but similar item B. These methods usually encounter major problems with large sparse matrices, since the number of user-item interactions can be too low for generating high quality clusters.

Model-based

These methods are based on machine learning and data mining techniques. The goal is to train models to be able to make predictions. For example, we could use existing user-item interactions to train a model to predict the top-5 items that a user might like the most. One advantage of these methods is that they are able to recommend a larger number of items to a larger number of users, compared to other methods like memory-based. We say they have large *coverage*, even when working with large sparse matrices.



(a) User-based filtering (b) Item-based filtering

Issues with collaborative filtering systems

There are two main challenges that come up with these systems:

1. Cold start: we should have enough information (user-item interactions) for the system to work. If we setup a new e-commerce site, we cannot give recommendations until users have interacted with a significant number of items.
2. Adding new users/items to the system: whether it is a new user or item, we have no prior information about them since they don't have existing interactions.

These problems can be alleviated by asking users for other type of data at the time of sign-up (gender, age, interests, etc), and using meta information from the items in order to be able to relate them to other existing items in the database.

3. Demographic Recommender Systems

In demographic recommender systems, the demographic information about the user is leveraged to learn classifiers that can map specific demographics to ratings or buying propensities. An early recommender system, referred to as Grundy, recommended books based on the library of manually assembled stereotypes. The characteristics of the user were collected with the use of an interactive dialogue. The work in observed that the demographic groups from marketing research can be used to recommend items. Another work makes Web page recommendations on the basis of the demographic characteristics of users that have rated a particular page highly. In many cases, demographic information can be combined with

additional context to guide the recommendation process. This approach is related to the methodology of context-sensitive recommender systems.

4. Knowledge-Based Recommender Systems

Knowledge-based recommender systems are particularly useful in the context of items that are not purchased very often. Examples include items such as real estate, automobiles, tourism requests, financial services, or expensive luxury goods. In such cases, sufficient ratings may not be available for the recommendation process. As the items are bought rarely, and with different types of detailed options, it is difficult to obtain a sufficient number of ratings for a specific instantiation (i.e., combination of options) of the item at hand.

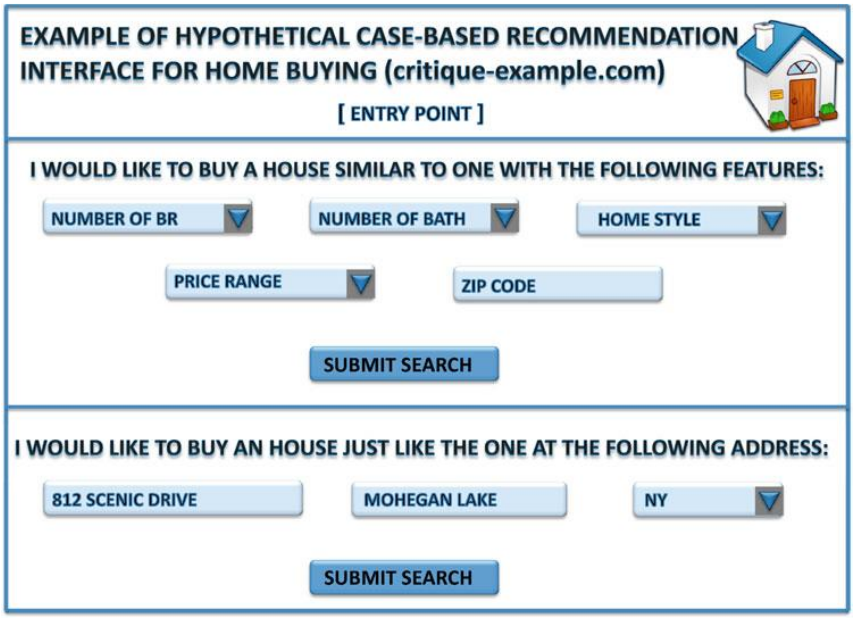
Knowledge-based recommender systems can be classified on the basis of the type of the interface (and corresponding knowledge) used to achieve the aforementioned goals:

1. *Constraint-based recommender systems:* In constraint-based systems, users typically specify requirements or constraints (e.g., lower or upper limits) on the item attributes. An example of such an interface is illustrated in Figure 1.5.

Domain-specific rules are used to match the user requirements to item attributes. These rules represent the domain-specific knowledge used by the system. Such rules could take the form of domain-specific constraints on the item attributes (e.g., "*Cars before year 1970 do not have cruise control.*"). Furthermore, constraint-based systems often create rules relating user attributes to item attributes (e.g., "*Older investors do not invest in ultra high-risk products.*"). In such cases, user attributes may also be specified in the search process. Depending on the number and type of returned results, the user might have an opportunity to modify their original requirements. For example, they might relax some of their constraints when too few results are returned, or they might add more constraints. This search process is interactively repeated until the user arrives at her desired results.

2. *Case-based recommender systems:* In case-based recommender systems, specific cases are specified by the user as targets or anchor points. Similarity metrics are defined on the item attributes to retrieve similar items to these cases. An example of such an interface is illustrated

in Figure 1.6. The similarity metrics are often carefully defined in a domain-specific way. Therefore, the similarity metrics form the domain knowledge that is used in such systems. The returned results are often used as new target cases with some interactive modifications by the user. For example, when a user sees a returned result, which is almost similar to what they want, they might re-issue a query with that target, but with some of the attributes changed to the user's liking. This interactive process is used to guide the user towards items of interest.



How is the interactivity in knowledge-based recommender systems achieved?

This guidance takes place through one or more of the following methods:

1. *Conversational systems:* In this case, the user preferences are determined iteratively in the context of a feedback loop. The main reason for this is that the item domain is complex and the user preferences can be determined only in the context of an iterative conversational system.

2. *Search-based systems:* In search-based systems, user preferences are elicited by using a preset sequence of questions such as the following: "Do you prefer a house in a suburban area or within the city?" In some cases, specific search interfaces may be set up in order to provide the ability to specify user constraints.

3. *Navigation-based recommendation:* In navigation-based recommendation, the user specifies a number of change requests to the item being currently recommended.

Through an iterative set of change requests, it is possible to arrive at a desirable item. An example of a change request specified by the user, when a specific house is being recommended is as follows: "I would like a similar house about 5 miles west of the currently recommended house." Such recommender systems are also referred to as ***critiquing recommender systems***

5. Hybrid and Ensemble-Based Recommender Systems

The three aforementioned systems exploit different sources of input, and they may work well in different scenarios. For example, collaborative filtering systems rely on community ratings, content-based methods rely on textual descriptions and the target user's own ratings, and knowledge-based systems rely on interactions with the user in the context of knowledge bases. Similarly, demographic systems use the demographic profiles of the users to make recommendations. It is noteworthy that these different systems use different types of input, and have different strengths and weaknesses. Some recommender systems, such as knowledge-based systems, are more effective in cold-start settings where a significant amount of data is not available. Other recommender systems, such as collaborative methods, are more effective when a lot of data is available.

In many cases where a wider variety of inputs is available, one has the flexibility of using different types of recommender systems for the same task. In such cases, many opportunities exist for hybridization, where the various aspects from different types of systems are combined to achieve the best of all worlds. Hybrid recommender systems are closely related to the field of ensemble analysis, in which the power of multiple types of machine learning algorithms is combined to create a more robust model. Ensemble-based recommender systems are able to combine not only the power of multiple data sources, but they are also able to improve the effectiveness of a particular class of recommender systems (e.g., collaborative systems) by combining multiple models of the same type.

The two primary models of Recommender systems are as follows:

1. *Prediction version of problem:* The first approach is to predict the rating value for a user-item combination. It is assumed that *training* data is available, indicating user preferences for items. For m users and n items, this corresponds to an incomplete $m \times n$ matrix, where the specified (or *observed*) values are used for training. The missing (or *unobserved*) values are predicted using this training model. This problem is also referred to as the *matrix completion problem* because we have an incompletely specified matrix of values, and the remaining values are predicted by the learning algorithm.

2. *Ranking version of problem:* In practice, it is not necessary to predict the ratings of users for specific items in order to make recommendations to users. Rather, a merchant may wish to recommend the top-k items for a particular user, or determine the top-k users to target for a particular item. The determination of the top-k items is more common than the determination of top-k users, although the methods in the two cases are exactly analogous. Throughout this book, we will discuss only the determination of the top-k items, because it is the more common setting. This problem is also referred to as the *top-k recommendation problem*, and it is the ranking formulation of the recommendation problem.

Common *operational* and *technical* goals of recommender systems

1. *Relevance:* The most obvious operational goal of a recommender system is to recommend items that are relevant to the user at hand. Users are more likely to consume items they find

interesting. Although relevance is the primary operational goal of a recommender system, it is not sufficient in isolation. Therefore, we discuss several secondary goals below, which are not quite as important as relevance but are nevertheless important enough to have a significant impact.

2. *Novelty:* Recommender systems are truly helpful when the recommended item is something that the user has not seen in the past. For example, popular movies of a preferred genre would rarely be novel to the user. Repeated recommendation of popular items can also lead to reduction in sales diversity.

3. *Serendipity:* A related notion is that of *serendipity*, wherein the items recommended are somewhat unexpected, and therefore there is a modest element of lucky discovery, as opposed to obvious recommendations. Serendipity is different from novelty in that the recommendations are truly *surprising* to the user, rather than simply something they did not know about before. It may often be the case that a particular user may only be consuming items of a specific type, although a latent interest in items of other types may exist which the user might themselves find surprising. Unlike novelty, serendipitous methods focus on discovering such recommendations. For example, if a new Indian restaurant opens in a neighborhood, then the recommendation of that restaurant to a user who normally eats Indian food is novel but not necessarily serendipitous. On the other hand, when the same user is recommended Ethiopian food, and it was unknown to the user that such food might appeal to her, then the recommendation is serendipitous. Serendipity has the beneficial side effect of increasing sales diversity or beginning a new trend of interest in the user. Increasing serendipity often has long-term and strategic benefits to the merchant because of the possibility of discovering entirely new areas of interest. On the other hand, algorithms that provide serendipitous recommendations often tend to recommend irrelevant items. In many cases, the longer term and strategic benefits of serendipitous methods outweigh these short-term disadvantages.

4. *Increasing recommendation diversity:* Recommender systems typically suggest a list of top-*k* items. When all these recommended items are very similar, it increases the risk that the user might not like *any* of these items. On the other hand, when the recommended list contains items of different types, there is a greater chance that the user might like at least one of these items. Diversity has the benefit of ensuring that the user does not get bored by repeated recommendation of similar items.

Popular examples of historical and current recommender systems

1. **GroupLens Recommender System -** GroupLens was a pioneering recommender system, which was built as a research prototype for recommendation of Usenet news. The system collected ratings from Usenet readers and used them to predict whether or not other readers would like an article before they read it. Some of the earliest automated collaborative filtering algorithms were developed in the GroupLens1 setting. The general ideas developed by this group were also extended to other product settings such as books and movies. The corresponding recommender systems were referred to as *BookLens* and *MovieLens*, respectively. Aside from its pioneering contributions to collaborative filtering research, the GroupLens research team was notable for releasing

several data sets during the early years of this field, when data sets were not easily available for benchmarking.

2. **Amazon.com Recommender System -** Amazon.com was also one of the pioneers in recommender systems, especially in the commercial setting. During the early years, it was one of the few retailers that had the foresight to realize the usefulness of this technology. Originally founded as a book e-retailer, the business expanded to virtually all forms of products. Consequently, Amazon.com now sells virtually all categories of products such as books, CDs, software, electronics, and so on.

   The recommendations in Amazon.com are provided on the basis of explicitly provided ratings, buying behavior, and browsing behavior. The ratings in Amazon.com are specified on a 5-point scale, with lowest rating being 1-star, and the highest rating being 5-star. The customer-specific buying and browsing data can be easily collected when users are logged in with an account authentication mechanism supported by Amazon. Recommendations are also provided to users on the main Web page of the site, whenever they log into their accounts. In many cases, explanations for recommendations are provided. For example, the relationship of a recommended item to previously purchased items may be included in the recommender system interface. The purchase or browsing behavior of a user can be viewed as a type of implicit rating, as opposed to an explicit rating, which is specified by the user. Many commercial systems allow the flexibility of providing recommendations both on the basis of explicit and implicit feedback.
3. **Netflix Movie Recommender System** - Netflix was founded as a mail-order digital video disc (DVD) rental company of movies and television shows, which was eventually expanded to streaming delivery. At the present time, the primary business of Netflix is that of providing streaming delivery of movies and television shows on a subscription basis. Netflix provides users the ability to rate the movies and television shows on a 5-point scale. Furthermore, the user actions in terms of watching various items are also stored by Netflix. These ratings and actions are then used by Netflix to make recommendations. Netflix does an excellent job of providing explanations for the recommended items. It explicitly provides examples of recommendations based on specific items that were watched by the user. Such information provides the user with additional information to decide whether or not to watch a specific movie. Presenting meaningful explanations is important to provide the user with an understanding of why they might find a particular movie interesting. This approach also makes it more likely for the user to act on the recommendation and truly improves the user experience. This type of interesting approach can also help improve customer loyalty and retention.
4. **Google News Personalization System -** The Google News personalization system is able to recommend news to users based on their history of clicks. The clicks are associated with specific users based on identification mechanisms enabled by Gmail accounts. In this case, news articles are treated as items. The act of a user clicking on a news article can be viewed as a positive rating for that article. Such ratings can be viewed as *unary ratings*, in which a mechanism exists for a user to express their affinity for an item, but no mechanism exists for them to show their dislike. Furthermore, the ratings are *implicit*, because they are *inferred* from user actions rather than being explicitly specified by the user. Nevertheless, variations of the approach can also be applied to cases where ratings are explicitly specified. Collaborative recommendation algorithms are applied to the collected ratings, so that inferences can be made about the personalized articles for specific users.

5. **Facebook Friend Recommendations -** Social networking sites often recommend potential friends to users in order to increase the number of social connections at the site. Facebook is one such example of a social networking Web site. This kind of recommendation has slightly different goals than a product recommendation. While a product recommendation directly increases the profit of the merchant by facilitating product sales, an increase in the number of social connections improves the experience of a user at a social network. This, in turn, encourages the growth of the social network. Social networks are heavily dependent on the growth of the network to increase their advertising revenues. Therefore, the recommendation of potential friends (or *links*) enables better growth and connectivity of the network. This problem is also referred to as *link prediction* in the field of social network analysis. Such forms of recommendations are based on *structural relationships* rather than ratings data. Therefore, the nature of the underlying algorithms is completely different.

PERSONALIZED AND NON – PERSONALIZED RECOMMENDER SYSTEMS

Recommender systems are tools that help users discover products, services, or content by predicting their preferences. They can be broadly categorized into two types: personalized and non-personalized recommender systems. Here's an overview of each:

Personalized Recommender Systems

Personalized recommender systems tailor recommendations based on the preferences, behavior, and interactions of individual users. These systems rely on data such as past purchases, browsing history, ratings, and demographic information to generate recommendations specific to each user.

Types of Personalized Recommender Systems:

1. **COLLABORATIVE FILTERING** is a popular technique used in recommender systems to predict a user's preferences by leveraging the preferences of other users. The fundamental idea behind collaborative filtering is that users who have agreed in the past will likely agree in the future. Therefore, if user A has a similar preference pattern to user B, then the items that B likes can be recommended to A, and vice versa.

Types of Collaborative Filtering

Collaborative Filtering can be broadly categorized into two types:

1. User-Based Collaborative Filtering

- **Description**: In user-based collaborative filtering, the system identifies users who are similar to the target user based on their past interactions (e.g., ratings, purchases). Once similar users are identified, the system recommends items that these users have liked but the target user hasn't yet interacted with.

- **How it Works**:
    1. Compute the similarity between users using a similarity measure like Pearson correlation or cosine similarity.
    2. Identify the top N users who are most similar to the target user.
    3. Aggregate the preferences of these similar users to generate a list of recommended items for the target user.
- **Example**: If two users, A and B, have rated several movies similarly, and B has watched a movie that A hasn't seen yet, that movie might be recommended to A.

2. Item-Based Collaborative Filtering

- **Description**: Item-based collaborative filtering flips the approach by focusing on the similarity between items rather than users. The system identifies items that are similar to the ones the target user has interacted with and recommends those items.
- **How it Works**:
    1. Compute the similarity between items using a similarity measure like cosine similarity or Pearson correlation.
    2. For each item that the target user has interacted with, identify similar items.
    3. Aggregate these similar items to generate a list of recommendations for the target user.
- **Example**: If a user has liked a particular book, the system might recommend other books that are frequently liked by users who also liked the original book.

Key Concepts in Collaborative Filtering

1. **Similarity Measures**:
    - **Cosine Similarity**: Measures the cosine of the angle between two vectors (e.g., user or item vectors), focusing on the orientation of the vectors rather than their magnitude.
    - **Pearson Correlation Coefficient**: Measures the linear correlation between two sets of data (e.g., user ratings), adjusting for their mean to account for differences in rating scales.
    - **Jaccard Similarity**: Measures the similarity between two sets by comparing the size of their intersection with the size of their union, often used for binary data.
2. **Neighborhood-Based Methods**:
    - **K-Nearest Neighbors (k-NN)**: Identifies the k most similar users (or items) to the target user (or item) and uses their preferences to make recommendations.

3. **Matrix Factorization**:
    - **Description**: A more advanced technique often used in collaborative filtering, particularly in large datasets like those found in Netflix or Amazon. Matrix factorization techniques (like Singular Value Decomposition (SVD)) decompose the user-item interaction matrix into lower-dimensional matrices that capture latent factors, which can be used to predict missing ratings or interactions.
    - **How it Works**: The user-item interaction matrix is factored into two matrices: one representing users and the other representing items. The dot product of these matrices approximates the original matrix, allowing for prediction of unknown values (e.g., ratings).
4. **Cold Start Problem**:
    - **Description**: Collaborative filtering systems often struggle with making recommendations for new users or new items, as there is little to no interaction data to base recommendations on. This is known as the cold start problem.
    - **Solutions**: Hybrid systems (combining collaborative filtering with content-based filtering), bootstrapping with initial data, and leveraging user demographics can help mitigate the cold start problem.

Advantages of Collaborative Filtering

- **No Domain Knowledge Required**: Unlike content-based filtering, which requires knowledge of item features, collaborative filtering relies solely on user interaction data.
- **Personalization**: Provides highly personalized recommendations by leveraging the behavior of similar users.

Challenges of Collaborative Filtering

- **Data Sparsity**: In large datasets, many users may have interacted with only a small fraction of available items, leading to a sparse user-item matrix, which can make it difficult to find similar users or items.
- **Scalability**: As the number of users and items grows, computing similarities and generating recommendations can become computationally expensive.
- **Cold Start Problem**: Difficulty in recommending items to new users or recommending new items due to the lack of interaction data.

Use Cases and Applications

- **E-commerce**: Recommending products based on the purchasing behavior of similar users (e.g., Amazon’s "Customers who bought this item also bought").

- **Streaming Services**: Suggesting movies or songs based on viewing or listening history (e.g., Netflix, Spotify).
- **Social Media**: Recommending friends or connections based on mutual connections or similar interests (e.g., Facebook, LinkedIn).

2. **CONTENT-BASED FILTERING** is a recommendation technique that focuses on recommending items similar to those a user has interacted with in the past, based on the content or attributes of those items. Unlike collaborative filtering, which relies on the behavior of other users, content-based filtering solely considers the characteristics of the items and the individual user's preferences.

How Content-Based Filtering Works

Content-based filtering involves the following steps:

1. **Item Representation**:
    - Items (e.g., movies, books, songs) are represented by a set of features or attributes. For example:
        - A movie might be represented by its genre, director, cast, and keywords.
        - A book might be represented by its genre, author, and keywords from its synopsis.
        - A song might be represented by its genre, artist, and tempo.
    - These features are typically encoded as a vector for each item.
2. **User Profile Creation**:
    - A user profile is created based on the features of the items the user has interacted with (e.g., items they've rated highly, purchased, or clicked on).
    - The user profile is also represented as a vector, often created by averaging the vectors of the items the user has liked.
3. **Similarity Calculation**:
    - The system calculates the similarity between the user profile vector and the vectors of other items in the catalog.
    - Common similarity measures include cosine similarity, Euclidean distance, and dot product.
    - Items that are most similar to the user profile are recommended to the user.

Key Concepts in Content-Based Filtering

1. **Feature Extraction**:
    - **Textual Data**: For items like books, movies, or articles, natural language processing (NLP) techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings can be used to extract and represent features.
    - **Multimedia Data**: For items like images, music, or videos, features might include visual elements, audio characteristics, or metadata like tags and categories.
2. **User Profiles**:
    - The user profile evolves over time as the user interacts with more items. The system continuously updates the profile to better reflect the user's current preferences.
3. **Similarity Measures**:
    - **Cosine Similarity**: Measures the cosine of the angle between the user profile vector and item vectors, focusing on their orientation.
    - **Euclidean Distance**: Measures the straight-line distance between vectors, capturing the magnitude of difference between the user profile and item features.
4. **Content-Based Learning**:
    - The system can learn a user's preferences more accurately over time as it gathers more data on the user's interactions.
    - Some advanced systems use machine learning models like Naive Bayes or decision trees to predict user preferences based on content features.

Advantages of Content-Based Filtering

- **Personalization**: Provides personalized recommendations based on the individual user's past behavior and preferences.
- **No Need for User Data**: Can make recommendations even with limited data on other users, as it does not rely on the behavior of others.
- **Transparency**: The reasons for recommendations are often clear (e.g., "Because you liked X, we recommend Y").

Challenges of Content-Based Filtering

- **Limited to Known Preferences**: The system can only recommend items similar to what the user has already interacted with, potentially leading to a "filter bubble" where users are only exposed to content similar to what they already know.

- **Feature Engineering**: Effectively representing items through features requires good domain knowledge and can be challenging, especially with complex items like movies or music.
- **Cold Start for New Items**: New items without sufficient features or descriptions may not be recommended effectively.

Use Cases and Applications

- **E-commerce**: Recommending products similar to those a user has viewed or purchased based on attributes like category, brand, and price (e.g., "You might also like...").
- **Streaming Services**: Suggesting content such as movies or songs based on genres, directors, or artists that a user frequently engages with (e.g., Netflix recommending similar movies).
- **News and Articles**: Providing article recommendations based on topics, keywords, and the reading history of the user (e.g., personalized news feeds).

Example

Consider a content-based recommendation system for a movie streaming service:

- **Item Representation**: Each movie is represented by a vector of features including genre, director, main actors, and keywords from its plot summary.
- **User Profile**: A user who has watched and liked several action movies directed by a specific director will have a profile vector that emphasizes those features.
- **Recommendation**: The system recommends new movies that have similar features, such as other action movies or films by the same director, to this user.

Content-based filtering is particularly powerful in domains where item attributes are rich and can be meaningfully related to user preferences. However, it's often combined with other techniques, such as collaborative filtering, to address its limitations and provide more diverse recommendations.

3. **HYBRID RECOMMENDER SYSTEMS** combine multiple recommendation techniques to improve the quality, accuracy, and diversity of recommendations. By leveraging the strengths of different approaches—such as collaborative filtering, content-based filtering, and others—hybrid systems can overcome the limitations of individual methods and provide more robust recommendations.

Why Use Hybrid Systems?

- **Overcome Limitations**: Different recommendation techniques have specific strengths and weaknesses. For example, collaborative filtering might struggle with the cold start problem, while content-based filtering might only recommend items similar to those the user already knows. Hybrid systems can balance these trade-offs.

- **Improve Accuracy**: By combining predictions from multiple models, hybrid systems can generate more accurate recommendations.
- **Enhance Diversity**: They can provide a broader range of recommendations, avoiding issues like the "filter bubble" where users only see content similar to what they've already interacted with.
- **Increase Robustness**: If one technique fails or provides poor results, another technique in the hybrid system can compensate.

Types of Hybrid Recommender Systems

There are several ways to combine different recommendation techniques in a hybrid system:

1. **Weighted Hybrid**
    - **Description**: Different recommendation techniques are combined by assigning weights to their outputs, and the final recommendation is a weighted sum of these outputs.
    - **Example**: A system might assign 70% weight to collaborative filtering and 30% weight to content-based filtering to generate the final recommendation.
    - **Use Case**: When you have confidence in the strengths of different techniques and want to combine them proportionally.
2. **Switching Hybrid**
    - **Description**: The system switches between different recommendation techniques based on certain conditions, such as the availability of user data or the type of items being recommended.
    - **Example**: If a user has a rich interaction history, collaborative filtering might be used; if not, the system switches to content-based filtering.
    - **Use Case**: In scenarios where the effectiveness of a technique depends on the context or specific user data.
3. **Cascade Hybrid**
    - **Description**: One recommendation technique is applied first, and its results are refined by another technique. The output from the first technique is fed into the second as input.
    - **Example**: A system might use collaborative filtering to generate a list of potential recommendations and then refine that list using content-based filtering to ensure the items match the user's interests.
    - **Use Case**: When you want to prioritize one technique but still incorporate another to improve the quality of recommendations.

4. **Feature Augmentation Hybrid**
    - **Description**: One recommendation technique generates features that are used as input to another technique. For example, content-based features might be used to enhance collaborative filtering.
    - **Example**: A system might use content-based filtering to generate additional features for items (e.g., genre, keywords) and then use these features in a collaborative filtering model to improve recommendations.
    - **Use Case**: When enhancing the input data for one model can improve the accuracy of another model.
5. **Meta-Level Hybrid**
    - **Description**: One recommendation model's output serves as the input to another model. This is similar to the cascade hybrid but typically involves more complex interactions between models.
    - **Example**: The user profile generated by a content-based filtering system might be used as input to a collaborative filtering model.
    - **Use Case**: When the relationship between different models is more intricate and one model's output provides a richer context for another.
6. **Ensemble-Based Hybrid**
    - **Description**: Similar to ensemble learning in machine learning, where multiple models are combined to improve prediction accuracy. Each recommendation technique is treated as a separate "model," and their predictions are combined.
    - **Example**: The system might generate recommendations using both collaborative filtering and content-based filtering independently, and then use a voting or averaging mechanism to decide the final recommendation.
    - **Use Case**: In systems where combining the strengths of multiple models can lead to better overall performance.

Example of a Hybrid Recommender System

Consider a streaming service that uses a hybrid recommender system combining collaborative filtering and content-based filtering:

- **Collaborative Filtering**: The system analyzes users' viewing habits to find similarities between users. It uses this information to recommend movies that similar users have enjoyed.
- **Content-Based Filtering**: The system also looks at the content features of movies (e.g., genre, director, cast) that a user has previously watched and liked, recommending movies with similar characteristics.
- **Hybrid Approach**:

- The system might first use collaborative filtering to generate a list of potential recommendations.
- Then, it applies content-based filtering to refine this list, ensuring that the recommended movies align with the user's specific content preferences.
- Alternatively, the system could combine the outputs of both approaches with a weighted sum to produce a final recommendation list.

Advantages of Hybrid Recommender Systems

- **Better Accuracy**: Combining multiple techniques can lead to more accurate recommendations.
- **Increased Diversity**: Users receive a broader range of recommendations, reducing the risk of overfitting to specific user preferences.
- **Mitigated Weaknesses**: The weaknesses of one technique can be offset by the strengths of another, leading to a more balanced system.
- **Robustness**: The system can perform well even when one of the models is not providing optimal results, as other models can compensate.

Challenges of Hybrid Recommender Systems

- **Complexity**: Hybrid systems can be more complex to design, implement, and maintain than single-technique systems.
- **Computational Cost**: Combining multiple models or techniques can increase the computational cost, particularly in real-time recommendation scenarios.
- **Tuning**: Finding the right balance or combination of different techniques requires careful tuning and experimentation.

Use Cases and Applications

- **E-commerce**: Retailers like Amazon use hybrid systems to combine collaborative filtering with content-based recommendations, helping to recommend products that are both popular among similar users and aligned with the user's past behavior.
- **Streaming Services**: Netflix uses a hybrid approach that combines collaborative filtering, content-based filtering, and other algorithms to suggest movies and shows.
- **Social Media**: Platforms like Facebook and LinkedIn may use hybrid systems to recommend friends or connections, combining the social graph (collaborative) with content interests (content-based).

Hybrid recommender systems represent the state-of-the-art in recommendation technology, offering the flexibility to deliver highly accurate and personalized recommendations across various domains.

Non-Personalized Recommender Systems

Non-personalized recommender systems provide the same set of recommendations to all users, regardless of individual preferences or behaviour. These systems typically rely on overall popularity, trends, or predefined rules to suggest items.

Types of Non-Personalized Recommender Systems:

1. Most Popular Items

- Description: Recommends the most popular items based on overall metrics like sales, views, or ratings. This approach assumes that popular items will have broad appeal.

- Example: A news website showcasing the most-read articles of the day to all visitors.

2. Trending Items

- Description: Recommends items that are currently trending or gaining popularity. This approach is time-sensitive and often used in platforms that value recency.

- Example: YouTube promoting viral videos or trending content.

3. Recently Added Items

- Description: Recommends the latest items added to a catalog, assuming that new content will attract attention.

- Example: A bookstore highlighting newly released books on its homepage.

4. Top-Rated Items

- Description: Recommends items that have received the highest ratings from users. This approach assumes that high-quality items will appeal to a broad audience.

- Example: An e-commerce site listing top-rated products in a specific category.

Advantages:

- Simplicity: Easy to implement and requires little to no user data.

- No Cold Start Problem: Works well even for new users who have no interaction history.

- Scalability: Can handle large numbers of users without significant performance issues.

Challenges:

- Lack of Personalization: Does not consider individual user preferences, leading to generic recommendations.

- Lower Engagement: Users may not find the recommendations relevant, reducing their likelihood of interaction.

- Overemphasis on Popularity: May perpetuate the popularity of already popular items, making it harder for niche or new items to gain visibility.

Use Cases and Applications

- Personalized Systems: Ideal for platforms with rich user interaction data, like streaming services (e.g., Netflix, Spotify), online retailers (e.g., Amazon), and social media platforms.

- Non-Personalized Systems: Suitable for scenarios where user data is scarce or where general trends are sufficient, such as news aggregators, public display systems, or new user experiences.

DATA MINING METHODS FOR RECOMMENDER SYSTEMS

Recommender Systems (RS) typically apply techniques and methodologies from other neighboring areas – such as Human Computer Interaction (HCI) or Information Retrieval (IR). However, most of these systems bear in their core an algorithm that can be understood as a particular instance of a Data Mining (DM) technique. The process of data mining typically consists of 3 steps, carried out in succession: Data Preprocessing, Data Analysis, and Result Interpretation

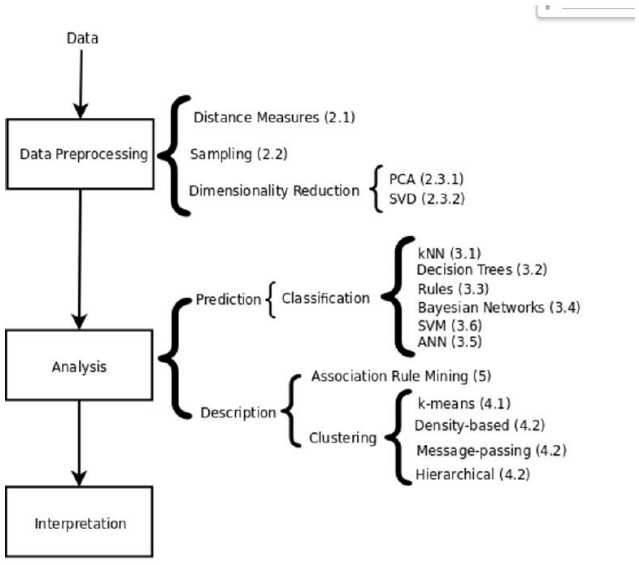


Fig: Main steps and methods in Data mining problem

Data Preprocessing

Data is defined as a collection of *objects* and their *attributes*, where an attribute is defined as a property or characteristic of an object. Other names for object include *record*, *item*, *point*, *sample*, *observation*, or *instance*.

An attribute might be also be referred to as a *variable*, *field*, *characteristic*, or *feature*. Real-life data typically needs to be *preprocessed* (*e.g.* cleansed, filtered, transformed) in order to be used by the machine learning techniques in the analysis step.

Sampling

Sampling is the main technique used in DM for selecting a subset of relevant data from a large data set. It is used both in the preprocessing and final data interpretation steps. Sampling may be used because processing the entire data set is computationally too expensive. It can also be used to create *training* and *testing* datasets.

The training dataset is used to learn the parameters or configure the algorithms used in the analysis step, while the testing dataset is used to evaluate the model or configuration obtained in the training phase, making sure that it performs well (*i.e.generalizes*) with previously unseen data.

The key issue to sampling is finding a subset of the original data set that is *representative*

– *i.e.* it has approximately the same property of interest – of the entire set.

Types of Sampling

Random sampling - where there is an equal probability of selecting any item.

Stratified sampling - the data is split into several partitions based on a particular feature, followed by random sampling on each partition independently.

sampling *without replacement* - When an item is selected, it is removed from the population.

sampling *with replacement* - where items are not removed from the population once they have been selected, allowing for the same sample to be selected more than once.

Sampling can lead to an **over-specialization** to the particular division of the training and testing data sets. For this reason, the training process may be repeated several times. The training and test sets are created from the original data set, the model is trained using the training data and tested with the examples in the test set.

The different training/test data sets are selected to start the training/testing process again that is repeated *K* times. Finally, the *average* performance of the *K* learned models is reported. This process is known as ***cross-validation***.

Types of cross validation techniques

In *repeated random sampling*, a standard random sampling process is carried out *K* times. In *n-Fold cross validation*, the data set is divided into *n* folds. One of the folds is used

for testing the model and the remaining $n-1$ folds are used for training. The cross validation process is then repeated n times with each of the n subsamples used exactly once as validation data.

The *leave-one-out (LOO)* approach can be seen as an extreme case of n-Fold cross validation where n is set to the number of items in the data set. Therefore, the algorithms are run as many times as data points using only one of them as a test each time.

SIMILARITY MEASURES

In recommender systems, similarity measures play a crucial role in determining how similar two items (like products, movies, etc.) or users are, which in turn helps in making personalized recommendations. Various similarity measures can be used depending on the type of data (e.g., ratings, user profiles, item features) and the recommendation technique (e.g., collaborative filtering, content-based filtering). Here are some commonly used similarity measures:

1. **Cosine Similarity**

Cosine similarity is a metric used to measure how similar two vectors are in terms of their orientation rather than their magnitude. It calculates the cosine of the angle between two non-zero vectors in a multi-dimensional space. This measure is often used in text analysis, information retrieval, and machine learning to compare the similarity between documents, sentences, or word embeddings.

Formula

The cosine similarity between two vectors **A** and **B** is calculated as:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Where:

- $A \cdot B$ is the dot product of the vectors A and B.
- $\|A\|$ is the magnitude (or length) of vector A.
- $\|B\|$ is the magnitude of vector B.

Interpretation

- The value of cosine similarity ranges between -1 and 1.
    - **1**: The vectors are identical, meaning the angle between them is 0°.
    - **0**: The vectors are orthogonal, meaning they have no similarity (the angle between them is 90°).
    - **-1**: The vectors are diametrically opposed, meaning they are completely dissimilar (the angle between them is 180°).

Applications

- **Text Analysis**: In natural language processing (NLP), cosine similarity is used to compare documents by representing them as vectors based on word frequencies or embeddings.
- **Recommender Systems**: It is used to find items similar to a user's preferences.
- **Clustering**: Cosine similarity can be used to group similar items or data points.

Cosine similarity is particularly useful when the magnitude of the vectors is not as important as the direction, making it a popular choice in various fields.

2. Pearson Correlation Coefficient

- **Description**: Measures the linear correlation between two variables (e.g., users' rating patterns). It accounts for the tendency of some users to give higher or lower ratings overall.

- Formula:

$$\text{Pearson Correlation}(A, B) = \frac{\sum(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum(A_i - \bar{A})^2 \sum(B_i - \bar{B})^2}}$$

Where A_i and B_i are the ratings given by users A and B to item i, and $\bar{A}$ and $\bar{B}$ are the average ratings of users A and B.

- Use Case: Collaborative filtering systems, particularly user-based collaborative filtering, where the similarity between users' rating patterns is key.

3. Jaccard Similarity

- **Description**: Measures similarity between two sets by comparing the size of their intersection with the size of their union.
- **Formula**:

$$\text{Jaccard Similarity}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Use Case: Useful in scenarios where the data is binary (e.g., users liking/disliking items, or items having/not having certain features).

4. Euclidean Distance

- **Description**: Measures the "straight-line" distance between two points in a multi-dimensional space.
- **Formula**:

$$\text{Euclidean Distance}(A, B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

Use Case: Can be used in content-based filtering to measure the similarity between items based on their features, though it's less common in collaborative filtering due to its sensitivity to magnitude.

5. Manhattan Distance (L1 Norm)

- **Description**: Measures the distance between two points along axes at right angles (i.e., the sum of the absolute differences of their coordinates).
- **Formula**:

$$\text{Manhattan Distance}(A, B) = \sum_{i=1}^{n} |A_i - B_i|$$

6. Hamming Distance

- **Description**: Measures the number of positions at which the corresponding elements of two binary vectors differ.
- **Use Case**: Used in scenarios with binary data (e.g., binary user-item interaction matrices) to measure similarity between users or items.

7. Adjusted Cosine Similarity

- **Description**: A variant of cosine similarity that adjusts for user bias by subtracting the user's average rating before computing the similarity.
- **Use Case**: User-based collaborative filtering, where it's important to account for different users' rating scales.

8. Tanimoto Coefficient

- **Description**: Similar to Jaccard similarity, but generalized for non-binary vectors.
- **Use Case**: Used in content-based filtering, particularly with non-binary feature vectors.

9. Dice Similarity Coefficient

- **Description**: Another measure for set similarity, similar to Jaccard, but gives more weight to the intersection.

Formula:

$$\text{Dice Similarity}(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|}$$

- **Use Case**: Used when the overlap (intersection) between sets is more significant for the application than the overall union size.

Applications in Recommender Systems:

- **User-based Collaborative Filtering**: Similarity measures like Pearson correlation, cosine similarity, and adjusted cosine similarity are used to find users with similar preferences.
- **Item-based Collaborative Filtering**: Measures such as cosine similarity and Jaccard similarity are used to find items similar to those the user has interacted with.
- **Content-based Filtering**: Similarity measures like cosine similarity and Euclidean distance are used to compare items based on their attributes or descriptions.

The choice of similarity measure can significantly impact the performance of a recommender system, and sometimes hybrid approaches combine multiple measures to improve accuracy.

Similarity measures for items with Binary attributes

Several similarity measures have been proposed in the case of items that only have binary attributes. First, the *M01*, *M10*, *M11*, and *M00* quantities are computed, where *M01* = the number of attributes where *x* was 0 and *y* was 1, *M10* = the number of attributes where *x* was 1 and *y* was 0, and so on. From those quantities we can compute:

The ***Simple Matching*** **coefficient** ***SMC*** $= \frac{number\ of\ matches}{numbero\ f\ attributes} = \frac{M11+M00}{M01+M10+M00+M11}$

The ***Jaccard*** **coefficient** ***JC*** $= \frac{M11}{M01+M10+M11}$

The ***Extended Jaccard (*****Tanimoto) coefficient**, a variation of JC for continuous or count attributes that is computed by $d = \frac{x \bullet y}{\|x\|2+\|x\|2-x \bullet y}$

REDUCING DIMENSIONALITY

It is common in RS to have not only a data set with features that define a high dimensional space, but also very sparse information in that space – *i.e.* there are values for a limited number of features per object. The notions of density and distance between points, which are critical for clustering and outlier detection, become less meaningful in highly dimensional spaces. This is known as the *Curse of Dimensionality*.

Dimensionality reduction techniques help overcome this problem by transforming the original high-dimensional space into a lower-dimensionality.

Sparsity and the *curse of dimensionality* are recurring problems in RS. Even in the simplest setting, we are likely to have a sparse matrix with thousands of rows and columns (*i.e.* users and items), most of which are zeros. Therefore, dimensionality reduction comes in naturally. Applying dimensionality reduction makes such a difference and its results are so directly applicable to the computation of the predicted value, that this is now considered to be an approach to RS design, rather than a preprocessing technique. the two most relevant dimensionality reduction algorithms in the context of RS: *Principal Component Analysis (PCA)* and *Singular Value Decomposition (SVD)*. These techniques can be used in isolation or as a preprocessing step for any of the other techniques

Singular Value Decomposition (SVD) is a powerful mathematical technique widely used in various fields, including data mining, machine learning, and recommender systems. It is particularly known for its effectiveness in reducing the dimensionality of data, capturing latent relationships, and compressing information.

Rank of a Matrix

The rank of a matrix is the maximum number of linearly independent row (or column) vectors in the matrix. A vector **r** is said to be linearly independent of vectors **r1** and **r2** if it cannot be expressed as a linear combination of **r1** and **r2**.

$$r \neq ar_1 + br_2$$

$$A = \begin{matrix} 1 & 3 & 7 \\ 2 & 6 & 14 \end{matrix} \qquad B = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 7 & 9 \end{matrix} \qquad C = \begin{matrix} 1 & 0 & 2 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \end{matrix}$$

Consider the three matrices below:

- In matrix A, row **r2** is a multiple of **r1**, ***r2 = 2 r1***, so it has only one independent row. **Rank(A) = 1**
- In matrix B, row **r3** is a sum of **r1** and **r2**, ***r3 = r1 + r2***, but **r1** and **r2** are independent. **Rank(B) = 2**
- In matrix C, all 3 rows are independent of each other. **Rank(C) = 3**

The rank of a matrix can be thought of as a representative of the amount of unique information represented by the matrix. Higher the rank, higher the information.

Singular Value Decomposition (SVD)

SVD deals with decomposing a matrix into a product of 3 matrices as shown:

$$A = USV^T$$

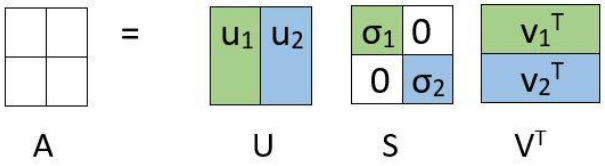
If the dimensions of A are m x n:

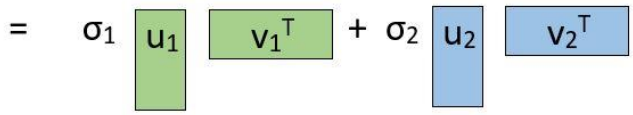
- U is an m x m matrix of **Left Singular Vectors**
- S is an m x n rectangular diagonal matrix of **Singular Values** arranged in decreasing order
- V is an n x n matrix of **Right Singular Vectors**

Why is SVD used in Dimensionality Reduction?

The reason can be understood by an alternate representation of the decomposition. See the figure below:

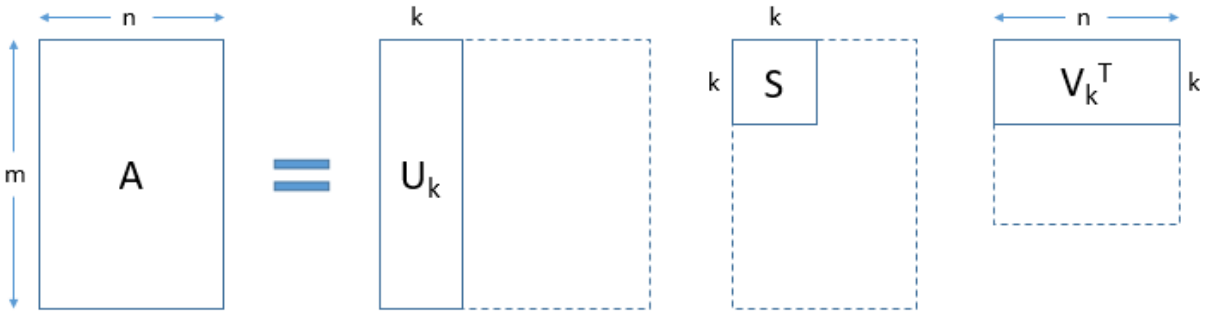
ISBN: 978-81-989720-3-3





The decomposition allows us to express our ***original matrix as a linear combination of low-rank matrices***.

In a practical application, you will observe that only the first few, say k, singular values are large. The rest of the singular values approach zero. As a result, terms except the first few can be ignored without losing much of the information. See how the matrices are truncated in the figure below:



To summarize:

- Using SVD, we are able to represent our large matrix A by 3 smaller matrices U, S and V
- This is helpful in large computations
- We can obtain a **k-rank approximation** of A. To do this, select the first k singular values and truncate the 3 matrices accordingly

3 Ways to Perform SVD in Python

Fortunately, we do not need to perform these calculations manually. We can implement SVD in Python in three simple ways.

SVD in NumPy

NumPy is the fundamental package for Scientific Computing in Python. It has useful Linear Algebra capabilities along with other applications.

You can obtain the complete matrices U, S, and V using SVD in **numpy.linalg**. Note that S is a diagonal matrix which means that most of its entries are zeros. This is called a **sparse matrix**. To save space, S is returned as a 1D array of singular values instead of the complete 2D matrix.

```
import numpy as np
from numpy.linalg import svd

# define your matrix as a 2D numpy array
A = np.array([[4, 0], [3, -5]])

U, S, VT = svd(A)

print("Left Singular Vectors:")
print(U)
print("Singular Values:")
print(np.diag(S))
print("Right Singular Vectors:")
print(VT)

# check that this is an exact decomposition
# @ is used for matrix multiplication in Py3, use np.matmul with Py2
print(U @ np.diag(S) @ VT)
```

Truncated SVD in scikit-learn

In most common applications, we do not want to find the complete matrices U, S and V. We are ultimately going to trim our matrices, so we do not want to find the complete matrices in the first place.

In such cases, it is better to use **TruncatedSVD** from **sklearn.decomposition**. You specify the number of features you want in the output as the **n_components** parameter. *n_components should be strictly less than the number of features in the input matrix*:

```
import numpy as np
from sklearn.decomposition import TruncatedSVD

A = np.array([[-1, 2, 0], [2, 0, -2], [0, -2, 1]])
print("Original Matrix:")
print(A)

svd =  TruncatedSVD(n_components = 2)
A_transf = svd.fit_transform(A)

print("Singular values:")
print(svd.singular_values_)

print("Transformed Matrix after reducing to 2 features:")
print(A_transf)
```

Randomized SVD in scikit-learn

Randomized SVD gives the same results as Truncated SVD and has a faster computation time. While Truncated SVD uses an exact solver ARPACK, Randomized SVD uses approximation techniques.

```
import numpy as np
from sklearn.utils.extmath import randomized_svd

A = np.array([[-1, 2, 0], [2, 0, -2], [0, -2, 1]])
u, s, vt = randomized_svd(A, n_components = 2)

print("Left Singular Vectors:")
print(u)

print("Singular Values:")
print(np.diag(s))

print("Right Singular Vectors:")
print(vt)
```

Applications of SVD in Recommender Systems

SVD is particularly useful in the context of collaborative filtering, where it helps in identifying latent factors in the user-item interaction matrix. Here's how SVD is applied:

1. **Data Representation:**
   - The user-item interaction matrix A is typically sparse, with many missing entries (e.g., users haven't rated many items). SVD can help fill in these missing values by identifying underlying patterns.
2. **Dimensionality Reduction:**
   - SVD reduces the dimensionality of the interaction matrix, capturing the most important features that describe the data. This reduction helps in overcoming the sparsity issue by focusing on the most significant latent factors.

3. **Prediction:**
   - By reconstructing the interaction matrix A from the reduced matrices U, Σ, and V, SVD can predict the missing entries. These predictions can be used to recommend items to users that they have not yet interacted with.
4. **Example in Recommender Systems:**
   - Suppose A represents a movie rating matrix where rows correspond to users and columns correspond to movies. Each entry in A represents a rating a user has given to a movie. SVD can decompose A into U, Σ, and V^T, where:
     - U represents the relationship between users and latent factors.
     - V^T represents the relationship between movies and latent factors.
     - Σ indicates the importance of each latent factor.
   - The reconstructed matrix approximates the original ratings, including predictions for missing entries, which can be used to recommend movies to users based on their inferred preferences.

Advantages of SVD:

- **Noise Reduction:** By focusing on the largest singular values, SVD filters out noise and emphasizes the most critical patterns.
- **Interpretability:** The latent factors identified by SVD can often be interpreted in meaningful ways, such as underlying genres or themes in a movie dataset.
- **Efficiency:** SVD enables efficient storage and computation by reducing the dimensionality of the data.

Challenges with SVD:

- **Scalability:** Computing SVD for very large matrices can be computationally expensive, though there are approximations (like truncated SVD) that address this.
- **Cold Start Problem:** Like other collaborative filtering techniques, SVD struggles with the cold start problem, where new users or items lack sufficient data for accurate predictions.

Singular Value Decomposition is a foundational technique in data mining and machine learning, with powerful applications in recommender systems. By uncovering the latent structure in data, SVD enables personalized recommendations, improves efficiency, and helps manage the complexity of large, sparse datasets. Its ability to reduce dimensionality while preserving essential information makes it a critical tool for developing effective recommendation algorithms.

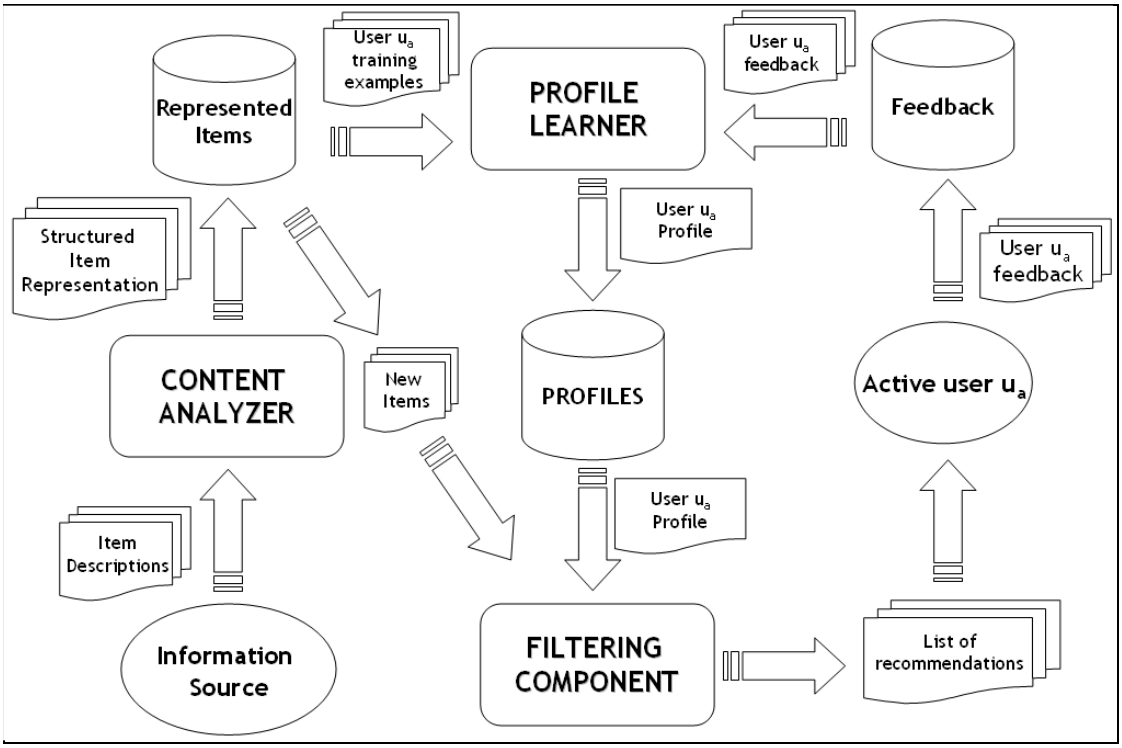
ISBN: 978-81-989720-3-3

UNIT II CONTENT-BASED RECOMMENDATION SYSTEMS

High-level architecture of content-based systems - Item profiles, Representing item profiles, Methods for learning user profiles, Similarity-based retrieval, and Classification algorithms.

2.1 HIGH-LEVEL ARCHITECTURE OF CONTENT-BASED SYSTEMS

Content-based Information Filtering (IF) systems need proper techniques for representing the items and producing the user profile, and some strategies for comparing the user profile with the item representation. The high-level architecture of a content-based recommender system is depicted in Figure 3.1. The recommendation process is performed in three steps, each of which is handled by a separate component:



CONTENT ANALYZER – When information has no structure (e.g. text), some kind of pre-processing step is needed to extract structured relevant information. The main responsibility of the component is to represent the content of items (e.g. documents, Web pages, news, product descriptions, etc.) coming from information sources in a form suitable for the next processing steps. Data items are analyzed by feature extraction techniques in order to shift item representation from the original information space to the target one (e.g.Web pages represented as keyword vectors).

Item descriptions coming from *Information Source* are processed by the CONTENT ANALYZER, that extracts features (keywords, n-grams, concepts, . . .) from unstructured text

to produce a structured item representation, stored in the repository *Represented Items*. This representation is the input to the PROFILE LEARNER and FILTERING COMPONENT

PROFILE LEARNER – This module collects data representative of the user preferences and tries to generalize this data, in order to construct the user profile. Usually, the generalization strategy is realized through machine learning techniques, which are able to infer a model of user interests starting from items liked or disliked in the past. For instance, the PROFILE LEARNER of a Web page recommender can implement a relevance feedback method in which the learning technique combines vectors of positive and negative examples into a prototype vector representing the user profile.

In order to construct and update the *profile* of the *active user* u_a (user for which recommendations must be provided) her reactions to items are collected in some way and recorded in the repository *Feedback (Annotations)*. These reactions together with the related item descriptions, are exploited during the process of learning a model useful to predict the actual relevance of newly presented items. Training examples are Web pages on which a positive or negative feedback has been provided by the user

Techniques to record user feedback

Explicit feedback - When a system requires the user to explicitly evaluate items, this technique is usually referred to as "explicit feedback"

Approaches to get explicit relevance feedback

- *like/dislike* – items are classified as "relevant" or "not relevant" by adopting a simple binary rating scale
- *ratings* – a discrete numeric scale is usually adopted to judge items. Alternatively, symbolic ratings are mapped to a numeric scale, where users have the possibility of rating a Web page as *hot*, *lukewarm*, or *cold*;
- *text comments* – Comments about a single item are collected and presented to the users as a means of facilitating the decision-making process, such as in. For instance, customer's feedback at Amazon.com or eBay.com might help users in deciding whether an item has been appreciated by the community.

Implicit feedback - does not require any active user involvement, in the sense that feedback is derived from monitoring and analyzing user's activities.

FILTERING COMPONENT – This module exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. The result is a binary or continuous relevance judgment (computed using some similarity metrics [42]), the latter case resulting in a ranked list of potentially interesting items. In the above-mentioned example, the matching is realized by computing the cosine similarity between the prototype vector and the item vectors.

Overview of Content based Recommendation process

Step 1: In order to build the profile of the active user u_a, the training set TR_a for u_a must be defined. TR_a is a set of pairs (I_k, r_k), where r_k is the rating provided by u_a on the item representation I_k.

Step 2: Given a set of item representation labeled with ratings, the PROFILE LEARNER applies supervised learning algorithms to generate a predictive model – the *user profile* – which is usually stored in a *profile repository* for later use by the FILTERING COMPONENT.

Step 3: Given a new item representation, the FILTERING COMPONENT predicts whether it is likely to be of interest for the active user, by comparing features in the item representation to those in the representation of user preferences (stored in the user profile). Usually, the FILTERING COMPONENT implements some strategies to rank potentially interesting items according to the relevance with respect to the user profile. Top-ranked items are included in a *list of recommendations* L_a, that is presented to u_a. User tastes usually change in time, therefore up-to-date information must be maintained and provided to the PROFILE LEARNER in order to automatically update the user profile.

Step 4: Further feedback is gathered on generated recommendations by letting users state their satisfaction or dissatisfaction with items in La.

Step 5: After gathering that feedback, the learning process is performed again on the new training set, and the resulting profile is adapted to the updated user interests. The iteration of the feedback-learning cycle over time allows the system to take into account the dynamic nature of user preferences.

Advantages and Drawbacks of Content-based Filtering

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

• USER INDEPENDENCE - Content-based recommenders exploit solely ratings provided by the active user to build her own profile. Instead, collaborative filtering methods need ratings from other users in order to find the "nearest neighbors" of the active user, i.e., users that have similar tastes since they rated the same items similarly. Then, only the items that are most liked by the neighbors of the active user will be recommended;

• TRANSPARENCY - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes liked that item;

• NEW ITEM - Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects

collaborative recommenders which rely solely on users' preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

Content-based systems have several shortcomings:

• LIMITED CONTENT ANALYSIS - Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like.

Both automatic and manually assignment of features to items could not be sufficient to define distinguishing aspects of items that turn out to be necessary for the elicitation of user interests.

OVER-SPECIALIZATION - Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called serendipity problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty. To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be rec-

ommended just that kind of movies. A "perfect" content-based technique would rarely find anything novel, limiting the range of applications for which it would be useful.

• NEW USER - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

Strategies for tackling the above mentioned problems:

1. Enhancing Content Representation:
    - Use of common-sense and domain-specific knowledge to improve content representation.
    - Features like WordNet and Wikipedia concepts provide more accurate and transparent item representation.
2. Integration of User-Defined Lexicons:
    - Incorporating folksonomies to account for evolving vocabularies in recommendation systems.

3. Serendipitous Recommendations:
    - Analyzing methods to recommend novel and interesting items to address the over-specialization problem.
4. Overcoming the New User Problem:
    - Utilizing social tags from user communities as feedback to generate recommendations when minimal or no ratings are available for new users.

State of the art of Content based Recommender Systems

1. **Content-Based Filtering**:
    - Uses the content of data items to predict relevance based on the **user's profile**.
    - Combines concepts from **Information Retrieval (IR)** and **Artificial Intelligence (AI)**.
2. **Role of Information Retrieval (IR)**:
    - Users seeking recommendations are engaged in an **information-seeking process**.
    - In IR systems, users provide a query (keywords), while in recommendation systems, the **user's profile** represents the information need.
    - Items like web pages, news, and emails require **document modeling techniques** due to their unstructured text content, unlike structured items with well-defined attributes.
3. **Role of Artificial Intelligence (AI)**:
    - Recommendation is treated as a **learning problem** that uses past user behavior.
    - **User profiles** can include keywords or rules reflecting long-term interests.
    - **Machine Learning (ML)** is used to:
        - Automatically learn user profiles.
        - Categorize new items based on user-labeled data as interesting or not.
        - Build predictive models to determine the relevance of new items.
4. **Advanced Techniques and Algorithms**:
    - Item representation techniques, including traditional text-based and advanced methods (e.g., ontologies, encyclopedic knowledge).
    - Algorithms suitable for these representation techniques.

2.2 Item Representation

Items that can be recommended to the user are represented by a set of features, also called attributes or properties. For example, in a movie recommendation application, features adopted to describe a movie are: actors, directors, genres, subject matter, . . .). When each item is described by the same set of attributes, and there is a known set of values the attributes may take, the item is represented by means of structured data. In this case, many ML algorithms can be used to learn a user profile.

In most content-based filtering systems, item descriptions are textual features extracted from Web pages, emails, news articles or product descriptions. Unlike structured data, there are no attributes with well-defined values. Textual features create a number of complications when learning a user profile, due to the natural language ambiguity.

The ***problem is that traditional keyword-based profiles*** are unable to capture the semantics of user interests because they are primarily driven by a string matching operation. If a string, or some morphological variant, is found in both the profile and the document, a match is made and the document is considered as relevant. ***String matching suffers from problems of***:

• POLYSEMY, the presence of multiple meanings for one word;

• SYNONYMY, multiple words with the same meaning.

The result is that, due to synonymy, relevant information can be missed if the profile does not contain the exact keywords in the documents while, due to polysemy, wrong documents could be deemed relevant. Semantic analysis and its integration in personalization models is one of the most innovative and interesting approaches proposed in literature to solve those problems. The key idea is the adoption of knowledge bases, such as lexicons or ontologies , for annotating items and representing profiles in order to obtain a “semantic” interpretation of the user information needs.

2.2.1 Keyword-based Vector Space Model

Most content-based recommender systems use relatively simple retrieval models, such as keyword matching or the Vector Space Model (VSM) with basic TF-IDF weighting.

Vector Space Model (VSM):

- Represents text documents in **n-dimensional space**, where:
    - Each dimension corresponds to a term in the vocabulary.
    - Each document is a vector of term weights, indicating the association between terms and the document.
- Document representation: Let D = {d1,d2,...,dN} denote a set of documents or corpus, and T = {t1,t2,...,tn} be the dictionary, that is to say the set of words in the corpus. T is obtained by applying some standard natural language processing

operations, such as tokenization, stop-words removal, and stemming. Each document d_j is represented as a vector in a n-dimensional vector space, so dj = $\{w_{1j}, w_{2j}, ..., d_{nj}\}$, where w_{kj} is the weight for term t_k in document d_j

.□ **Dictionary (T)**:

- Created using **Natural Language Processing (NLP)** techniques:
  - Tokenization
  - Stop-word removal
  - Stemming

□ **Key Issues in VSM**:

- **Term Weighting**: Assigning appropriate weights to terms.
- **Feature Vector Similarity**: Measuring similarity between vectors.

□ **TF-IDF Weighting**:

- Based on three assumptions:
  1. **IDF (Inverse Document Frequency)**: Rare terms are not less relevant than frequent terms.
  2. **TF (Term Frequency)**: Multiple occurrences of a term in a document are more relevant than single occurrences.
  3. **Normalization**: Prevents long documents from being favored over short ones.

□ **TF-IDF Explanation**:

- **TF (Term Frequency)**: Measures how often a term occurs in a document.
- **IDF (Inverse Document Frequency)**: Weighs terms based on their rarity in the entire corpus.
- **Normalization**: Ensures fairness between short and long documents.

Terms that are frequent in one document but rare in the corpus are more relevant to the document's topic. In addition, normalizing the resulting weight vectors prevent longer documents from having a better chance of retrieval. These assumptions are well exemplified by the TF-IDF function:

$$\text{TF-IDF}(t_k, d_j) = \underbrace{\text{TF}(t_k, d_j)}_{\text{TF}} \cdot \underbrace{log\frac{N}{n_k}}_{\text{IDF}} \quad \text{----------} \quad 2.1$$

where N denotes the number of documents in the corpus, and n_k denotes the number of documents in the collection in which the term t_k occurs at least once.

$$\mathrm{TF}(t_k, d_j) = \frac{f_{k,j}}{max_z f_{z,j}} \quad \text{---------- 2.2}$$

where the maximum is computed over the frequencies fz, j of all terms tz that occur in document dj. In order for the weights to fall in the [0,1] interval and for the documents to be represented by vectors of equal length, weights obtained by Equation (3.1) are usually normalized by cosine normalization:

$$w_{k,j} = \frac{\text{TF-IDF}(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} \text{TF-IDF}(t_s, d_j)^2}} \quad \text{---------- 2.3}$$

which enforces the normalization assumption. As stated earlier, a similarity measure is required to determine the closeness between two documents. Many similarity measures have been derived to describe the proximity of two vectors; among those measures, cosine similarity is the most widely used:

$$sim(d_i, d_j) = \frac{\sum_k w_{ki} \cdot w_{kj}}{\sqrt{\sum_k w_{ki}^2} \cdot \sqrt{\sum_k w_{kj}^2}} \quad \text{---------- 2.4}$$

In content-based recommender systems relying on VSM, both user profiles and items are represented as weighted term vectors. Predictions of a user's interest in a particular item can be derived by computing the cosine similarity.

2.2.2 Keyword-Based Recommender Systems

1. **Overview**:
    - Keyword-based systems represent items and user profiles using **keyword vectors**.
    - Used in applications like **news, music, e-commerce, movies**, etc.
    - Focus is on identifying terms or keywords to represent user preferences and item content.

Examples of Systems in Various Domains

A. Web Browsing:

- **Amalthea**:
    - Assists users in finding relevant pages using **filtering agents**.
    - Pages are represented as **weighted vectors**.
    - Visited pages = positive examples; Non-visited = negative examples.

- **Syskill & Webert**:
    - Represents documents with **128 most informative words** (informativeness determined via multiple methods).
- **ifWeb**:
    - Profiles represented as **weighted semantic networks** with **explicit feedback**.
    - Features:
        - Tracks both interests and disinterests.
        - Incorporates **temporal decay**, i.e., older interests lose relevance.
- **WebMate**:
    - Tracks user interests across domains.
    - User profile = multiple keyword vectors, representing **independent interests**.

B. News Filtering:

- **NewT (News Tailor)**:
    - Users provide **positive/negative feedback** on articles, parts of articles, authors, or sources.
    - Multiple filtering agents trained for different news categories (e.g., sports, politics).
- **YourNews**:
    - Maintains **short-term profiles** (20 most recent articles) and **long-term profiles** (all past views).
    - Topics covered: National, World, Business, etc.
    - Represents user profiles using **weighted prototype term vectors** extracted from past views.
- **NewsDude**:
    - Builds:
        - **Short-term model** using TF-IDF and cosine similarity.
        - **Long-term model** using Naïve Bayes, trained with user feedback on interesting articles.
    - News source: Yahoo! News.

- **Daily Learner**:
    - Uses **Nearest Neighbor classification** for short-term interests.
    - Uses **Naïve Bayes** for long-term interests, collecting data over extended periods.

C. Advanced Representations:

- **PSUN**:
    - Uses **n-grams** (patterns of recurring words) to represent articles.
    - Profiles compete using **genetic algorithms**.
- **INFOrmer**:
    - Articles and user profiles represented as **semantic networks**.
    - Adapts to user interests using **spreading activation techniques** that propagate relevance through connected nodes.

D. Other Application Domains:

- **Books (LIBRA)**:
    - Uses **Naïve Bayes categorization** for book recommendations.
    - Extracts product descriptions from Amazon pages.
- **Emails (Re:Agent)**:
    - Automates actions like filtering, forwarding, etc., based on training from user-classified folders.
- **Movies**:
    - **INTIMATE**:
        - Learns preferences from movie synopses (Internet Movie Database).
        - Users rate movies into categories (e.g., excellent, bad).
    - **Movies2GO**:
        - Combines **voting schemes** to manage conflicting preferences within a single user.
- **Music**:
    - **Pandora**:
        - Manual annotation of music for recommendations (scalability issue).
    - **FOAFing the Music**:

- Combines:
  - **Friend of a Friend (FOAF)** descriptions.
  - Context from music-related RSS feeds.
  - Automatically extracted audio content.

Limitations of Keyword-Based Systems

1. **Performance depends on training data**:
   - Systems rely on sufficient **positive/negative examples** for training.
   - Provides accurate recommendations when training data is large and diverse.
2. **Lack of intelligence**:
   - Focuses on exact **keyword matching**.
   - Example: For a user interested in "French Impressionism," keyword-based systems will only return results with the exact words **French** and **Impressionism**.
   - Misses related content, e.g., **Claude Monet** or **Renoir exhibitions**, even though they are highly relevant.
3. **Need for advanced semantic techniques**:
   - To address limitations, systems require **semantic intelligence** to analyze relationships beyond keywords.
   - Techniques like **ontologies** and **encyclopedic knowledge** can enrich content representation.

Hybrid Recommender Systems

- Combine **collaborative filtering** and **content-based approaches** to improve recommendations.
- Examples:
  - **Fab, WebWatcher, P-Tango, ProfBuilder, PTV, Content-boosted Collaborative Filtering, CinemaScreen**.

1. Keyword-based systems are effective with **large training datasets** and sufficient user evidence.
2. Systems excel in domains where keywords capture **clear user interests**.
3. However, for **complex or abstract interests**, advanced techniques like **semantic analysis** are necessary to improve accuracy and relevance.

2.2.3 **Semantic Analysis Using Ontologies**

Semantic analysis enhances the quality of user profiles in recommender systems by leveraging external knowledge sources, such as **ontologies** and **lexical databases**. These resources bring in cultural, linguistic, and domain-specific background knowledge to interpret and reason about content more accurately.

Key Elements of Ontology-Based Semantic Analysis

1. **Knowledge Source**
    - **Lexical databases**: Example: WordNet, MultiWordNet.
    - **Ontologies**: Domain-specific (e.g., IPTC ontology for news, DMOZ for academic topics).
2. **Techniques for Annotation and Representation**
    - **Word Sense Disambiguation (WSD)**: Distinguishes between different meanings of words to enhance document representation.
    - **Word Domain Disambiguation**: Maps content to specific semantic domains using external knowledge.
    - **Semantic annotations**: Automatic or semi-automatic processes that associate text with ontology concepts or synsets.
3. **User Profile Content**
    - Profiles enriched with **semantic concepts**, e.g., synsets, categories, or ontology-based features.
    - Content tailored to user preferences by leveraging hierarchical relationships in ontologies.
4. **Item-Profile Matching**
    - **Similarity measures**: Semantic similarity using cosine similarity or spreading activation.
    - **Reasoning techniques**: Infer additional knowledge from structured profiles (e.g., OWL ontologies).

Examples of Semantic Systems and Their Features

1. SiteIF

- Domain: Multilingual news websites.
- Knowledge Source: MultiWordNet (English-Italian lexical database).
- Item Representation: News articles annotated with synsets using Word Domain Disambiguation.

- User Profile: Semantic network of synsets representing user interests.
- Matching Strategy: Semantic Network Value Technique estimates document relevance by comparing synsets.

2. ITR (Item Recommender)

- Domain: Movies, books, music.
- Knowledge Source: WordNet lexical ontology.
- Item Representation:
    - Bag-of-Synsets (BOS): Synset-based extension of the Bag-of-Words model.
- User Profile: Naïve Bayes binary classifier identifies important synsets indicating user preferences.
- Matching Strategy: Probabilistic scoring for item relevance based on user profile synsets.

3. SEWeP (Semantic Enhancement for Web Personalization)

- Domain: Web page personalization.
- Knowledge Source: Domain-specific taxonomy of categories and WordNet.
- Item Representation: Keywords mapped to taxonomy categories using WordNet similarity measures.
- Matching Strategy: Patterns in navigational data are matched with categories of interest to suggest additional pages.

4. Quickstep

- Domain: Academic research papers.
- Knowledge Source: DMOZ directory ontology.
- Item Representation: Papers classified into 27 topics using a k-Nearest Neighbor classifier.
- User Profile: Topics correlated with previously browsed papers and associated interest scores.
- Matching Strategy: Matches top topics in the user profile with newly classified papers.

5. Informed Recommender

- Domain: Consumer product reviews.
- Knowledge Source: Translation ontology formalizing product and opinion quality.

- Item Representation: Text mining converts review content into structured ontology-based information.
- Matching Strategy: Recommendations are based on user queries about specific product features.

6. News@hand

- Domain: News recommendation system.
- Knowledge Source: Adapted IPTC ontology covering multiple domains (e.g., politics, science).
- Item Representation: Articles represented as **TF-IDF vectors** in a concept space defined by ontologies.
- User Profile: Captures user interest intensity in different ontology-defined concepts.
- Matching Strategy: Cosine similarity between user profiles and article vectors.

7. Interactive Digital TV

- Domain: TV program recommendation.
- Knowledge Source: OWL ontology for the TV domain.
- Item Representation: Metadata describing program attributes enriched with semantic annotations.
- User Profile: Ontology-based representation of preferences that enables reasoning.
    - : Spreading activation explores semantic relationships between user preferences and programs.

Innovative Semantic Features in Recommendation Systems

1. **JUMP System**
    - Handles complex queries for knowledge workers, such as "Prepare a report for the VIKEF project."
    - Semantic matching between queries and documents using a domain ontology.
    - Processes relations (e.g., "technical report" and "project") to identify relevant content.
2. **Integration of Ontologies and Text Mining**
    - Ontologies structure and formalize user and item representations.
    - Text mining automates annotation, e.g., extracting product opinions or mapping news to concepts.

Benefits of Ontology-Based Systems

1. **Improved Personalization**
    - Semantic context improves recommendation accuracy compared to keyword-based methods.
2. **Flexibility**
    - Ontology-based reasoning uncovers implicit relationships and broadens the scope of recommendations.
3. **Cross-Domain Adaptability**
    - Recommender systems like News@hand demonstrate how multiple domain ontologies can handle diverse user interests.
4. **Enhanced User Experience**
    - Systems like Quickstep and Foxtrot offer visualizations of profiles, helping users better understand recommendations.

Challenges and Research Directions

1. **WordNet Limitations**
    - WordNet is insufficient for domain-specific nuances, emphasizing the need for specialized ontologies.
2. **Scalability**
    - Large-scale systems require efficient processing for ontology-based annotation and reasoning.
3. **Automation**
    - Manual taxonomy building (e.g., SEWeP) is labor-intensive and demands more robust automatic annotation methods.
4. **Hybrid Approaches**
    - Combining linguistic ontologies (e.g., WordNet) with domain-specific knowledge improves user contextualization.
5. **Real-Time Processing**
    - Dynamic user interests require adaptive systems that leverage semantic knowledge in real-time.

Ontology-based semantic analysis is transforming recommender systems by incorporating structured knowledge, enabling a deeper understanding of content and user preferences. By bridging linguistic and domain-specific resources, these systems achieve

greater personalization, adaptability, and relevance. However, challenges such as scalability, automation, and domain specificity drive ongoing research to refine semantic recommender methodologies.

2.2.4 Semantic Analysis Using Encyclopedic Knowledge Sources

1. **Why Use Encyclopedic Knowledge?**
    - Traditional text analysis techniques, such as the **bag-of-words (BOW)** model, often lack semantic depth.
    - Incorporating **common-sense** and **domain-specific knowledge** (exogenous knowledge) helps create richer, more meaningful representations compared to solely extracting information from text (endogenous knowledge).
2. **Examples of Encyclopedic Knowledge Sources:**
    - **Wikipedia**: Widely used for its breadth, accuracy, multilingual support, and constant updates.
    - **Open Directory Project (ODP)** and **Yahoo! Web Directory**: Other general-purpose knowledge bases.

Key Techniques for Semantic Enrichment

1. **Explicit Semantic Analysis (ESA):**
    - Represents texts as vectors in a **high-dimensional space of Wikipedia concepts**.
        - Example concepts: *ITALY, COMPUTER SCIENCE, RECOMMENDER SYSTEMS.*
    - **Advantages:**
        - Provides fine-grained semantic representations.
        - Outperforms traditional methods in **word/text relatedness** and **text categorization** tasks.
        - Enhances **retrieval models** when combined with BOW.
2. **Wikify! System:**
    - **Keyword Extraction**: Identifies significant concepts in a text.
    - **Word Sense Disambiguation**: Links concepts to corresponding Wikipedia articles, enriching text with semantically related annotations.
    - **Evaluation**:

        - A Turing-like test revealed that human and system-generated annotations are nearly indistinguishable.
    - **Applications**: Enriching documents with world knowledge for better semantic understanding.

Applications in Recommender Systems

1. **Current State:**
    - Despite its potential, **advanced semantic representations** (e.g., ESA, Wikify!) have not yet been extensively applied to **content-based recommender systems**.
    - Existing systems rely heavily on collaborative filtering or shallow representations of user profiles.
2. **Research Example:**
    - Wikipedia was used to calculate **movie similarities** for the Netflix Prize competition.
        - Methods: Leveraged Wikipedia's content and hyperlink structure to generate a similarity matrix.
        - Algorithms: Combined **k-Nearest Neighbors** and **Pseudo-SVD**.
        - **Outcome**: Despite innovative use of semantic knowledge, there was no significant improvement in prediction accuracy.

Opportunities for Future Research

1. **Exploration of Hybrid Models:**
    - Combine encyclopedic knowledge with collaborative filtering or other recommendation techniques.
    - Use semantic profiles to enhance personalization in recommendations.
2. **Application to User Profiles:**
    - Develop systems that build user profiles enriched with world facts, enabling more accurate and context-aware recommendations.
3. **Improving Prediction Accuracy:**
    - Experiment with different algorithms for integrating encyclopedic knowledge, such as deep learning techniques.

- Explore the use of **ontology-based reasoning** and **hierarchical concept modeling**.

4. **New Use Cases:**
    - Incorporate encyclopedic knowledge into emerging applications like **healthcare**, **education**, and **e-commerce** recommendations.

Semantic analysis using encyclopedic knowledge sources like Wikipedia has shown promise in enriching text representations and improving tasks like categorization and retrieval. However, its application to learning user profiles and recommender systems remains underexplored, presenting a fertile area for future innovation.

2.4 Methods for Learning User Profiles

Learning user profiles is a key task in content-based recommender systems, where the goal is to model individual user preferences. This process can be formulated as a **binary text categorization task**. Below is an outline of the methodology and techniques involved:

Problem Definition

- **Binary Categorization Task:**
    - Each document is classified into one of two categories:
        - **Positive Class (c+):** Represents content the user likes.
        - **Negative Class (c−):** Represents content the user dislikes.
    - User profiles are learned by identifying features associated with user preferences.

General Approach

1. **Inductive Learning:**
    - Uses labeled training data (documents with known relevance scores).
    - Builds a **text classifier** to identify patterns that separate interesting documents from uninteresting ones.
2. **Key Steps:**
    - **Document Labeling:** Users assign relevance scores to documents.
    - **Feature Extraction:** Identify and extract key features (e.g., words, phrases, topics) representing the documents.
    - **Model Training:** Train a learning algorithm to distinguish between c+ and c− based on the extracted features.

- **Profile Creation:** The trained model serves as a representation of the user's preferences.
- **Document Ranking:** Apply the model to rank or filter new documents based on user preferences.

Popular Machine Learning Algorithms for User Profile Learning *(extra topic)*

1. **Naive Bayes Classifier:**
   - Probabilistic approach that assumes features are independent given the class.
   - **Advantages:**
     - Simple and computationally efficient.
     - Effective with small datasets.
   - **Applications:** Text categorization, spam filtering, etc.
2. **k-Nearest Neighbors (k-NN):**
   - Non-parametric method that classifies a document based on the labels of its closest neighbors in feature space.
   - **Advantages:**
     - Easy to implement and interpret.
     - Works well for high-dimensional data.
   - **Limitations:** Computationally expensive for large datasets.
3. **Support Vector Machines (SVM):**
   - Finds a hyperplane that best separates the positive and negative classes in feature space.
   - **Advantages:**
     - Effective in high-dimensional spaces.
     - Robust to overfitting in sparse data.
   - **Applications:** Widely used for text categorization and content filtering.
4. **Decision Trees:**
   - Builds a tree structure based on feature splits to classify documents.
   - **Advantages:**
     - Intuitive and interpretable.

        - Can handle both categorical and numerical features.
    - **Limitation:** Prone to overfitting without proper pruning.

5. **Random Forest:**
    - An ensemble of decision trees that improves classification accuracy through majority voting.
    - **Advantages:**
        - Reduces overfitting compared to single decision trees.
        - Handles high-dimensional data well.
6. **Neural Networks:**
    - Use deep learning architectures to model complex relationships between features and preferences.
    - **Advantages:**
        - Captures non-linear patterns and hierarchical representations.
        - Scales well with large datasets.
    - **Applications:** Advanced recommendation systems, image, and text classification.
7. **Logistic Regression:**
    - Models the probability that a document belongs to the positive class.
    - **Advantages:**
        - Simple and effective for binary classification tasks.
        - Well-suited for interpretable results.

Advantages of Machine Learning Approaches

- Personalization: Learns individual user preferences over time.
- Automation: Reduces manual effort by automatically inferring profiles.
- Adaptability: Models can adapt to changes in user behavior or preferences.

Applications in Content-Based Recommender Systems

- Document Filtering: Rank new documents based on their predicted relevance.
- Media Recommendations: Suggest articles, books, or movies tailored to user interests.
- E-Learning Systems: Recommend study materials based on learner profiles.

Challenges

1. **Cold Start Problem:**
    - Learning accurate profiles is difficult with limited user feedback.
2. **Scalability:**
    - Handling large datasets or high-dimensional feature spaces can be computationally intensive.
3. **Feature Selection:**
    - Identifying the most informative features to represent user preferences effectively.
4. **Evolving Preferences:**
    - Adapting to changes in user interests over time.

2.4.1 Probabilistic Methods and Naïve Bayes in User Profiling

Naïve Bayes is a probabilistic method from the family of **Bayesian classifiers**. It estimates the likelihood of a document belonging to a specific class using probabilities derived from training data. It is widely used for text classification, including learning user profiles in **content-based recommender systems**, where the task is to classify documents (e.g., articles, movies) as relevant or irrelevant based on user preferences.

1. Theoretical Foundation: Bayes' Theorem - Bayes' Theorem Formula:

$$P(c|d) = \frac{P(c) \cdot P(d|c)}{P(d)}$$

Where:

- $P(c|d)$: Posterior probability (probability of class c given document d).
- $P(c)$: Prior probability (probability of class c independent of d).
- $P(d|c)$: Likelihood (probability of document d occurring in class c).
- $P(d)$: Evidence (probability of observing d across all classes).

Practical Simplification: The denominator P(d) is constant for all classes and can be ignored during classification. Therefore:

$$P(c|d) \propto P(c) \cdot P(d|c)$$

The document is assigned to the class with the **highest posterior probability**:

$$c = \arg\max_{c_j} P(c_j) \cdot P(d|c_j)$$

2. Naïve Bayes Assumptions

1. **Conditional Independence**:
    - Assumes all words (or tokens) in a document are conditionally independent of each other given the class.
    - Simplifies probability estimation to individual words rather than entire documents.
    - This assumption is often violated in real-world data but still works well empirically.
2. **Feature Representation**:
    - Treats documents as feature vectors over the vocabulary of the corpus.
    - Ignores word order and context, focusing solely on word occurrence or frequency.

3. Event Models in Naïve Bayes

Naïve Bayes uses two main event models for text representation:

a. Multivariate Bernoulli Model

- Represents a document as a binary vector:
    - Each word is marked as **present (1)** or **absent (0)** in the document.
- Suitable for smaller vocabularies or datasets where word frequency is less important.
- Loses information about word repetition within a document.

b. Multinomial Model

- Represents a document as a word frequency vector:
    - Counts the number of times each word appears in the document.
- Captures the importance of word frequency.
- Superior performance, especially with **large vocabularies**.

Multinomial Model Formula:

$$P(c_j|d_i) \propto P(c_j) \prod_{t_k \in V_{d_i}} P(t_k|c_j)^{N(d_i,t_k)}$$

Where:

- V_{d_i}: Vocabulary subset (words present in document d_i).
- $N(d_i, t_k)$: Frequency of word t_k in document d_i.
- $P(t_k|c_j)$: Probability of word t_k given class c_j.

4. Smoothing Techniques

Smoothing adjusts probabilities to avoid overfitting or assigning zero probabilities to unseen words.

a. Laplace Smoothing:

- Adds a constant (usually 1) to all word counts.
- Ensures all probabilities are non-zero.

b. Witten-Bell Smoothing:

- Balances probabilities for observed and unobserved events.
- More sophisticated than Laplace and adapts better to sparse d

5. Advantages of Naïve Bayes

1. **Efficiency**:
    - Computationally fast, even for large datasets.
    - Works well in real-time applications.
2. **Simplicity**:
    - Easy to implement with minimal parameter tuning.
    - Requires fewer resources compared to complex algorithms like SVM.
3. **Effectiveness for Sparse Data**:
    - Handles high-dimensional text data effectively.
    - Performs well despite independence assumptions.
4. **Robustness**:
    - Works reliably even with small training datasets or violations of independence assumptions.

6. Limitations

1. **Independence Assumption**:
    - Words are not truly independent in real-world text.
    - Reduces performance in contexts where word relationships matter.

2. **Word Order Ignored**:
    - Cannot capture semantic nuances from word arrangement.
3. **Imbalanced Classes**:
    - Performance is affected when training data has more positive feedback (c^+) than negative feedback (c^-).
4. **Performance on Rare Categories**:
    - Struggles with underrepresented classes due to limited training examples.
5. **Document Length Variation**:
    - Rough parameter estimation when documents have widely varying lengths.

7. Challenges in User Profiling

1. **Feedback Imbalance**:
    - Users often provide more positive feedback (liked items) than negative feedback (disliked items), leading to class imbalance.
2. **Training Data Issues**:
    - Sparse training datasets for negative examples make it difficult to model user dislikes (c^-).
3. **Document Length Variability**:
    - Inconsistent document lengths affect probability estimates.

8. Adaptations for User Profiling

1. **Multivariate Poisson Model**:
    - Extends Naïve Bayes to better estimate parameters for user profiling tasks.
    - More effective for datasets with variable document lengths and imbalanced classes.
2. **Class-Specific Weighting**:
    - Emphasizes better representation of the minority class (c^-) by assigning weights or using synthetic data generation.

9. Applications in Recommender Systems

1. **Content-Based Recommendations**:
    - Learn user preferences (c^+) to rank or recommend relevant items.
    - Filter out irrelevant items based on inferred dislikes (c^-).
2. **Cold Start Problem**:
    - Efficiently works with limited data to generate initial recommendations.
3. **Text-Based Categorization**:
    - Classifies and ranks documents, articles, or media items based on user interests.

Naïve Bayes is a foundational method in probabilistic text classification. While its assumptions may not fully align with real-world data, it remains highly effective and efficient for user profiling tasks. With adaptations like smoothing and advanced probabilistic models, it continues to play a significant role in recommender systems and natural language processing.

2.4.2. Rocchio's Algorithm: A Profile-Based Classifier

2.1. General Idea

Rocchio's Algorithm is a popular adaptation of relevance feedback for inducing **linear profile-style classifiers** in content-based recommender systems. It works by:

- Representing documents as vectors in a multidimensional space.
- Combining document vectors (positive and negative examples) to create a **prototype vector** for each category.

The algorithm is **linear** and uses the **TF-IDF term weighting scheme** to compute vector components:

- **TF (Term Frequency):** Number of times a term appears in a document.
- **IDF (Inverse Document Frequency):** Importance of a term across all documents.

2.2. Rocchio's Formula

The classifier for category c_i, represented as a vector $\vec{c}_i = \langle \omega_{1i}, \omega_{2i}, \ldots, \omega_{|T|i} \rangle$, is computed as:

$$\omega_{ki} = \beta \cdot \frac{\sum_{d_j \in POS_i} \omega_{kj}}{|POS_i|} - \gamma \cdot \frac{\sum_{d_j \in NEG_i} \omega_{kj}}{|NEG_i|}$$

Where:

- ω_{kj}: TF-IDF weight of term t_k in document d_j.
- POS_i: Set of positive examples (relevant documents) for category c_i.
- NEG_i: Set of negative examples (irrelevant documents) for category c_i.
- $|POS_i|, |NEG_i|$: Number of positive and negative examples.
- β, γ: Parameters controlling the relative importance of positive and negative examples.

2.3. Document Classification

To classify a new document d:

1. Represent d as a vector $\vec{d}$.
2. Compute **similarity** between $\vec{d}$ and each prototype vector $\vec{c_i}$:
   - Similarity is often measured using **cosine similarity**:

$$\text{cosine similarity} = \frac{\vec{c_i} \cdot \vec{d}}{||\vec{c_i}||\,||\vec{d}||}$$

3. Assign d to the class c_i with the highest similarity value.

2.4. Parameters in Rocchio's Algorithm

- Controls the weight of positive feedback (relevant documents). A larger β emphasizes positive examples.
- γ: Controls the weight of negative feedback (irrelevant documents). A larger γ emphasizes penalizing negative examples.
- Default values: Commonly, β = 1, γ = 0.5, but these can be tuned based on the dataset.

2.5. Strengths of Rocchio's Algorithm

1. **Efficiency**:
   - Computationally simple and fast due to its linear nature.
2. **Interpretability**:
   - The prototype vector explicitly represents the category profile.
3. **Incremental Learning**:
   - Easily updated with new relevance feedback.

4. **Foundation in Relevance Feedback**:
    - Integrates directly with user feedback for dynamic profile refinement.

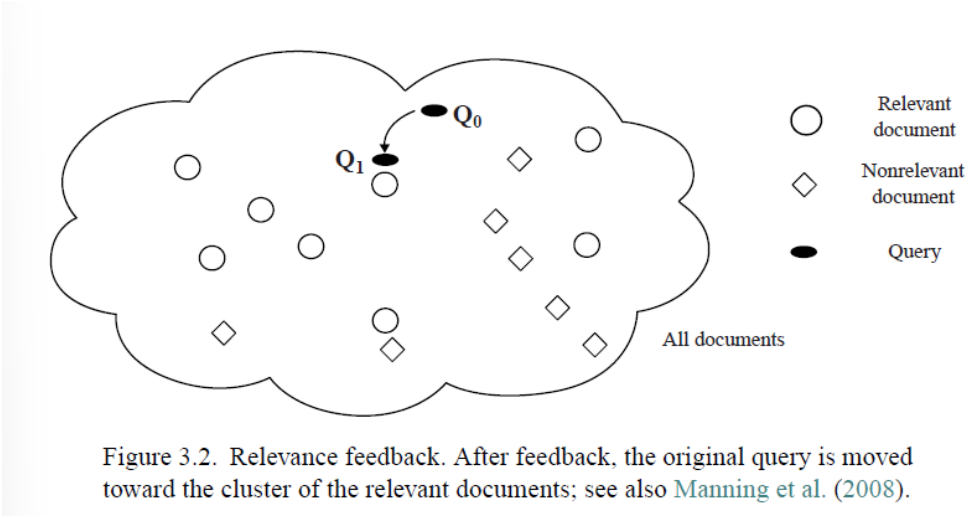


Figure 3.2. Relevance feedback. After feedback, the original query is moved toward the cluster of the relevant documents; see also Manning et al. (2008).

2.6. Limitations

1. **No Theoretical Guarantees**:
    - Lacks a solid theoretical foundation for performance or convergence.
2. **Sensitivity to Feedback Quality**:
    - Performance depends on the quality and quantity of feedback (positive vs. negative examples).
3. **Overemphasis on Negative Feedback**:
    - If negative feedback is dominant, the profile vector may deviate too far from relevant categories.
4. **Static Weighting**:
    - The choice of β\betaβ and γ\gammaγ is not adaptive to varying datasets.

2.7. Applications in Content-Based Recommender Systems

Rocchio's algorithm has been widely used to refine user profiles in recommender systems. For example:

- **YourNews**: A content-based news recommendation system where user feedback on articles (liked or disliked) is used to refine the recommendation model dynamically.

2.8. How Rocchio's Algorithm Works in Practice

1. **Initial User Profile**:
    - The system starts with an initial user profile based on prior interactions or general preferences.
2. **Feedback Loop**:
    - Users rate recommended documents as **relevant** or **irrelevant**.
    - These ratings modify the prototype vector by emphasizing positive feedback and penalizing negative feedback.
3. **Dynamic Updates**:
    - With each interaction, the system updates the user profile to better reflect their preferences.

2.9. Comparison to Other Methods

- **Naïve Bayes**:
    - Probabilistic and effective for imbalanced data.
    - Rocchio focuses more on vector similarity than probability estimation.
- **SVM (Support Vector Machines)**:
    - Non-linear and performs better on complex datasets.
    - Rocchio is simpler and computationally cheaper.
- **Collaborative Filtering**:
    - Focuses on user-user or item-item relationships.
    - Rocchio is purely content-based and does not consider other users.

Rocchio's Algorithm is a linear classifier well-suited for **content-based recommendation systems**. It leverages relevance feedback to refine user profiles dynamically, making it effective for personalized recommendations. Despite its simplicity and lack of theoretical guarantees, it remains a practical choice for systems requiring fast and interpretable profile updates.

Other Methods in Content-Based Recommendation Systems

Several learning algorithms have been applied in content-based recommender systems. Below is a brief overview of the key methods:

1. **Decision Trees**
    - Decision trees are a type of classifier where internal nodes are labeled by terms, and branches represent tests based on the weight of the term in the document.
    - The tree is built by recursively partitioning the data (text documents) into subgroups, ensuring that each subgroup contains only one class.
    - The partition criterion is typically based on information gain or entropy.
    - Example: Used in the **Syskill & Webert** recommender system.
2. **Decision Rule Classifiers**
    - These classifiers work similarly to decision trees, using recursive data partitioning.
    - The main advantage is that decision rules are often more compact than decision trees.
    - Rule learners try to select the best rule(s) that cover the training examples based on minimality criteria.
3. **Nearest Neighbor Algorithms (k-NN)**
    - These are "lazy learners" that store all training data in memory and classify new items based on similarity to stored items.
    - The class label is determined by the labels of the nearest neighbors (e.g., using cosine similarity in the vector space model).
    - The drawback of nearest neighbor algorithms is inefficiency during classification, as there is no true training phase, and all computation is deferred to classification time.
    - Example: Used in **Daily Learner** and **Quickstep** for modeling short-term user interests and associating semantic annotations with class names, respectively.

Similarity-Based Retrieval in Content-Based Recommender Systems

In content-based recommendation systems, the primary objective is to suggest items that are similar to those the user has shown interest in, based on the profile generated from previous interactions. The key task in such systems is to predict whether a user will like an item they have not yet interacted with. This is generally achieved by evaluating the similarity between unseen items and those the user has liked or interacted with in the past. **Similarity-based retrieval** forms the core approach of content-based recommendation systems.

1. Nearest Neighbor Approach (kNN)

The nearest neighbor method is a simple and intuitive approach to content-based recommendation systems. It predicts the user's potential interest in unseen items by evaluating how similar they are to items the user has rated or interacted with previously. The main components involved in this method are:

a) User Feedback

- Explicit Feedback: This is feedback provided directly by the user, such as likes, ratings, or comments.
- Implicit Feedback: This refers to feedback inferred from user behavior, such as clicks, time spent on an item, or viewing patterns.

b) Document Similarity

- Vector Space Model (VSM): Items (documents) are represented as vectors in a multi-dimensional space, where each dimension corresponds to a specific term (word).
- Cosine Similarity: The most commonly used measure for calculating the similarity between two document vectors. It computes the cosine of the angle between two vectors. The closer the cosine value is to 1, the more similar the documents are.

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|}$$

Where A and B are the vectors of the two documents, and $\|A\|$ and $\|B\|$ are their magnitudes.

c) Prediction Process

- To predict the relevance of an item that the user has not yet seen, the system first identifies the **k most similar items** that the user has rated or interacted with.
- These items are used to "vote" on the new item. For example, if a user liked 4 out of the 5 most similar items, the system predicts that the user will likely like the unseen item as well.
- **Voting Mechanisms**:
  - Binary voting: Items are either liked or disliked, and the vote is based on whether the item was liked or not.
  - Weighted voting: The votes from items are weighted according to how similar they are to the target item (using cosine similarity, for example).

d) Varying Parameters in kNN

- Neighborhood size (k): The number of nearest neighbors to consider. A small value of k might result in noisy predictions, whereas a large value could cause the system to generalize too much.
- Binarization of ratings: Ratings are often converted to binary values, i.e., liked or disliked. This reduces complexity but may lose some information.
- Minimum similarity threshold: A threshold that defines how similar items must be to each other for one to be considered in the neighbor set.
- Weighted voting: The influence of each neighbor's vote is weighted by how similar the neighbor is to the item in question. Neighbors that are more similar will have a greater influence on the final recommendation.

e) Example: Mobile News Recommendation System

- The **Billsus et al. (2000)** news recommendation system uses the kNN method to provide personalized news recommendations. This system adapts to a user's short-term interests by focusing on items the user has recently rated. When a new item arrives, the system compares it to the recent items the user has interacted with and makes recommendations based on the similarity between those items.

2. Short-term and Long-term User Profiles

A recommender system needs to balance short-term interests (e.g., current news stories) and long-term interests (e.g., general topics of interest).

a) Short-term Profile

- This model focuses on a user's recent activity. The idea is that users often have changing preferences based on their current interests (e.g., recent events, trending topics).
- In a short-term profile, only recent interactions or ratings are considered, and the system dynamically updates the user profile to focus on these interests.
- **Example**: For news recommendations, a user might be interested in following the latest updates about a specific event, which could change day-to-day.

b) Long-term Profile

- The long-term profile is a more stable model that tracks a user's preferences over an extended period. It is useful for identifying persistent interests and general themes that the user prefers.
- **TF-IDF** (Term Frequency-Inverse Document Frequency) is commonly used to identify the most important terms in documents over a long period. Terms that consistently

appear with high TF-IDF scores in documents that the user has liked are deemed important for the user's long-term profile.

c) Combining Short-term and Long-term Models

- A common challenge is how to balance these two types of models. The short-term profile is more dynamic and responsive to recent preferences, while the long-term profile helps capture the user's broader interests.
- **Combination Strategies**:
    - Priority to short-term profile: If the user has recent ratings, the system prioritizes these ratings in generating recommendations.
    - Fallback to long-term profile: If there is not enough recent data (e.g., the user has not rated anything recently), the system uses the long-term profile.
    - Hybrid models: Some systems combine short-term and long-term profiles and update the user model accordingly, possibly by weighting the short-term preferences more heavily when they are available.

3. Advantages of kNN-Based Methods

- Simple and Easy to Implement: The kNN algorithm is straightforward and does not require complex model training, making it easy to implement.
- Adaptation to Recent Changes: kNN can quickly adapt to changes in the user's behavior or preferences since it relies on the most recent data.
- Minimal Training Data: Unlike other algorithms, kNN can function well with a relatively small amount of user feedback or ratings.

4. Limitations of kNN

- Scalability: As the number of users and items increases, the computation required for determining the nearest neighbors increases significantly. The lack of a true training phase means that every classification requires calculating the similarity between the test item and all the training items.
- Prediction Accuracy: Pure kNN methods can suffer from lower prediction accuracy compared to more sophisticated algorithms such as decision trees, SVMs, or matrix factorization methods, especially when the number of neighbors is not optimally chosen.
- Sparsity: In many real-world scenarios, user-item interactions are sparse (i.e., users have rated only a small fraction of the items), making it difficult for kNN to make accurate predictions for users with limited history.

5. Real-world Applications

- Personalized News Recommendations: As demonstrated by **Billsus et al. (2000)**, kNN can be used to model a user's short-term interests in the news, allowing the system to make highly relevant recommendations based on recent user behavior.
- Music Recommender Systems: kNN has been used in music recommendation systems, where the system predicts songs the user might like based on the similarity to songs they have listened to before.

The **k-nearest neighbor (kNN)** approach is an essential and simple technique for content-based recommender systems. It predicts user preferences by comparing new items with items the user has previously interacted with, based on a measure of similarity (like cosine similarity). While kNN is easy to implement and adapts quickly to changes in user preferences, its main drawbacks are inefficiency at scale and lower prediction accuracy compared to more sophisticated algorithms. However, it remains a valuable method in scenarios where user preferences are dynamic and require frequent updates based on short-term interactions.

2.6 CLASSIFICATION ALGORITHMS

Linear Models in Classification:

- In a recommendation system, distinguishing between relevant and non-relevant items (e.g., documents or products) can be framed as a classification problem.
- Linear classifiers aim to find coefficients {w1,w2,…,wn} and a bias term b that define a decision boundary. This decision boundary separates the two classes (e.g., relevant and non-relevant).

Two-Dimensional Representation:

- For simplicity, consider a 2D feature space, where each document is represented as a point with coordinates (x1,x2).
- The classifier's decision boundary in this space is a line defined by the equation: $w1x1+w2x2= b$
- Documents on one side of the line are classified as relevant, and those on the other side as non-relevant.

Generalization to Higher Dimensions:

- In multidimensional feature spaces, the decision boundary is no longer a line but a **hyperplane**. The equation of the hyperplane is:

$$\mathbf{w} \cdot \mathbf{x} = b$$

where $\mathbf{w} = [w_1, w_2, \ldots, w_n]$ is the weight vector, $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is the feature vector, and b is the bias term.

- **Other Linear Classifiers:**

- Beyond basic linear classifiers, various machine learning algorithms employ linear decision boundaries. Examples include:
  - **Logistic Regression:** Uses a probabilistic approach and models the decision boundary as a sigmoid function applied to a linear combination of features.
  - **Support Vector Machines (SVM):** Finds the hyperplane that maximizes the margin between the two classes.
  - **Perceptrons:** A basic neural network model for linear classification.
  - **Linear Discriminant Analysis (LDA):** Projects the data onto a line and separates classes based on maximizing the distance between means relative to within-class variance.

Applications in Content-Based Recommendations:

- Linear models are suitable when the relationship between features and relevance is linear or can be approximated as such.
- These classifiers often serve as baseline models, and non-linear methods (e.g., kernel-based SVM, neural networks) can be used when the data exhibits non-linear patterns.

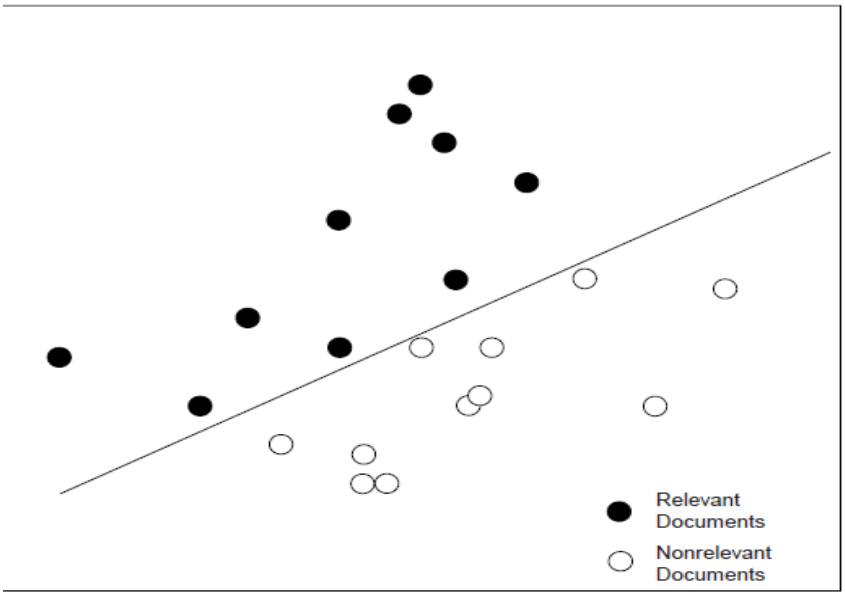


Figure: A linear classifier in two-dimensional space

The classification of an individual document is based on checking whether for a certain document $w1x1 + w2x2 > b$, which can be done very efficiently. In n-dimensional space, a generalized equation using weight and feature vectors instead of only two values is used, so the classification function is $\vec{w}^T \vec{x} = b$.

Linear Classifiers in Text Classification

1. **Linear Classifiers Overview:**
    - Many text classification algorithms, like **naive Bayes** and the **Rocchio method**, are linear classifiers (Manning et al., 2008).
    - Other linear classifiers include:
        - Widrow-Hoff algorithm (Widrow and Stearns, 1985).
        - Support Vector Machines (SVMs) (Joachims, 1998).
    - k-Nearest Neighbors (kNN) is **not** a linear classifier.
2. **Hyperplanes and Decision Boundaries:**
    - Infinite hyperplanes (or lines in 2D) can separate the document space.
    - Different learning methods identify different hyperplanes, affecting classification accuracy on test data.
    - **SVMs** maximize the margin (distance to data points) for better accuracy.
3. **Challenges with Linear Classifiers:**
    - **Noise in data**:
        - Noisy features: Features that mislead the classifier.
        - Noise documents: Misplaced documents that don't fit their cluster.
    - Handling noise is non-trivial and requires careful data preprocessing.
4. **Comparative Performance:**
    - Studies (Lewis et al., 1996; Yang and Liu, 1999) show **SVM-based methods** often outperform others, but no single method is best for all scenarios.
    - Non-linear classifiers may be needed when classification borders cannot be approximated by a line or hyperplane.
5. **Key Insight:**
    - Selecting the right learning method is essential for solving text classification problems (Manning et al., 2008).

ISBN: 978-81-989720-3-3

UNIT III COLLABORATIVE FILTERING

A systematic approach, Nearest-neighbor collaborative filtering (CF), user-based and item-based CF, components of neighborhood methods (rating normalization, similarity weight computation, and neighborhood selection

3.1 Formal definition of Item Recommendation task

The set of users in the system will be denoted by U, and the set of items by I. We denote by R the set of ratings recorded in the system, and write S the set of possible values for a rating (e.g., S = [1,5] or S = {like, dislike}). Also, we suppose that no more than one rating can be made by any user u ∈ U for a particular item i ∈ I and write r_{ui} this rating.

To identify the subset of users that have rated an item i, we use the notation Ui. Likewise, I_u represents the subset of items that have been rated by a user u. Finally, the items that have been rated by two users u and v, i.e. $I_u \cap I_v$, is an important concept in our presentation, and we use I_{uv} to denote this concept. In a similar fashion, U_{ij} is used to denote the set of users that have rated both items i and j.

3.2 Collaborative filtering approaches

Unlike content-based approaches, which use the content of items previously rated by a user *u*, collaborative (or social) filtering approaches rely on the ratings of *u* as well as those of other users in the system. The key idea is that the rating of *u* for a new item *i* is likely to be similar to that of another user *v*, if *u* and *v* have rated other items in a similar way. Likewise, *u* is likely to rate two items *i* and *j* in a similar fashion, if other users have given similar ratings to these two items.

Limitations of Content-based system overcome by Collaborative approaches

- Items for which the content is not available or difficult to obtain can still be recommended to users through the feedback of other users.
- Collaborative recommendations are based on the quality of items as evaluated by peers, instead of relying on content that may be a bad indicator of quality.
- Unlike content-based systems, collaborative filtering ones can recommend items with very different content, as long as other users have already shown interest for these different items.

3.3 Collaborative filtering methods

I. **Neighbourhood based or heuristic based collaborative filtering** - the user-item ratings stored in the system are directly used to predict ratings for new items.

This can be done in two ways:

1. ***User-based recommendation*** - evaluate the interest of a user *u* for an item *I* using the ratings for this item by other users, called *neighbors*, that have similar rating patterns.

The neighbors of user *u* are typically the users *v* whose ratings on the items rated by both *u* and *v*, i.e. *Iuv*, are most correlated to those of *u*. Eg: GroupLens, Bellcore video, Ringo

2. ***Item-based recommendation -*** predict the rating of a user u for an item i based on the ratings of u for items similar to i. In such approaches, two items are similar if several users of the system have rated these items in a similar fashion.

II. Model based collaborative filtering –

Model-based approaches use these ratings to learn a predictive model. The general idea is to model the user-item interactions with factors representing latent characteristics of the users and items in the system, like the preference class of users and the category class of items. This model is then trained using the available data, and later used to predict ratings of users for new items.

Eg: Bayesian Clustering, Latent Semantic Analysis, Latent Dirichlet Allocation, Maximum Entropy, Boltzmann Machines, Support Vector Machines, and Singular Value Decomposition

3.4 Advantages of neighborhood-based methods

- **Simplicity:** Neighborhood-based methods are intuitive and relatively simple to implement. In their simplest form, only one parameter (the number of neighbors used in the prediction) requires tuning.
- **Justifiability:** Such methods also provide a concise and intuitive justification for the computed predictions. For example, in item-based recommendation, the list of neighbor items, as well as the ratings given by the user to these items, can be presented to the user as a justification for the recommendation. This can help the user better understand the recommendation and its relevance, and could serve as basis for an interactive system where users can select the neighbors for which a greater importance should be given in the recommendation.
- **Efficiency:** One of the strong points of neighborhood-based systems is their efficiency. Unlike most model-based systems, they require no costly training phases, which need to be carried out at frequent intervals in large commercial applications. While the recommendation phase is usually more expensive than for model-based methods, the nearest-neighbors can be pre-computed in an offline step, providing near instantaneous recommendations. Moreover, storing these nearest neighbors requires very little memory, making such approaches scalable to applications having millions of users and items.
- **Stability:** Another useful property of recommender systems based on this approach is that they are little affected by the constant addition of users, items and ratings, which are typically observed in large commercial applications. For instance, once item similarities have been computed, an item-based system can readily make recommendations to new users, without having to re-train the system. Moreover, once a few ratings have been entered for a new item, only the similarities between this item and the ones already in the system need to be computed.

ISBN: 978-81-989720-3-3

3.5 Neighborhood-based Recommendation – User based and Item based CF

Recommender systems based on nearest-neighbors automate the common principle of *word-of-mouth*, where one relies on the opinion of like-minded people or other trusted sources to evaluate the value of an item (movie, book, articles, album, etc.) according to his own preferences. To illustrate this, consider the following example based on the ratings of Figure

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2	?	3	5	4
Diane	4	3	5	3	

Figure 1: An example showing the rating of four users for five movies

Example 1: User Eric has to decide whether or not to rent the movie "Titanic" that he has not yet seen. He knows that Lucy has very similar tastes when it comes to movies, as both of them hated "The Matrix" and loved "Forrest Gump", so he asks her opinion on this movie. On the other hand, Eric finds out he and Diane have different tastes, Diane likes action movies while he does not, and he discards her opinion or considers the opposite in his decision.

3.5.1 User-based Rating Prediction

User-based neighborhood recommendation methods predict the rating r_{ui} of a user u for a new item i using the ratings given to i by users most similar to u, called nearest-neighbors. Suppose we have for each user $v \neq u$ a value w_{uv} representing the preference similarity between u and v (how this similarity can be computed will be discussed later).

The k-nearest-neighbors (k-NN) of u, denoted by $N(u)$, are the k users v with the highest similarity w_{uv} to u. However, only the users who have rated item i can be used in the prediction of r_{ui}, and we instead consider the k users most similar to u that have rated i. We write this set of neighbors as $Ni(u)$. The rating r_{ui} can be estimated as the average rating given to i by these neighbors:

$$\hat{r}_{ui} = \frac{1}{|\mathcal{N}_i(u)|} \sum_{v \in \mathcal{N}_i(u)} r_{vi}. \qquad \text{---------} \quad 3.1$$

A problem with (3.1) is that is does not take into account the fact that the neighbors can have different levels of similarity. Consider once more the example of Figure 3.1. If the two nearest-neighbors of Eric are Lucy and Diane, it would be foolish to consider equally their ratings of the movie "Titanic", since Lucy's tastes are much closer to Eric's than Diane's.

A common solution to this problem is to weigh the contribution of each neighbor by its similarity to u. However, if these weights do not sum to 1, the predicted ratings can be well outside the range of allowed values. Consequently, it is customary to normalize these weights, such that the predicted rating becomes

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \quad \text{---------- 3.2}$$

In the denominator of (3.2), $|w_{uv}|$ is used instead of w_{uv} because negative weights can produce ratings outside the allowed range.

Equation (3.2) also has an important flaw: it does not consider the fact that users may use different rating values to quantify the same level of appreciation for an item. For example, one user may give the highest rating value to only a few outstanding items, while a less difficult one may give this value to most of the items he likes. This problem is usually addressed by converting the neighbors' ratings r_{vi} to normalized ones $h(r_{vi})$, giving the following prediction:

$$\hat{r}_{ui} = h^{-1}\left(\frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} h(r_{vi})}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}\right). \quad \text{---------- 3.3}$$

3.5.2 User-based Classification

Neighborhood-based *classification*, finds the most likely rating given by a user u to an item i, by having the nearest-neighbors of u vote on this value. The vote v_{ir} given by the k-NN of u for the rating $r \in S$ can be obtained as the sum of the similarity weights of neighbors that have given this rating to i:

$$v_{ir} = \sum_{v \in \mathcal{N}_i(u)} \delta(r_{vi} = r)\, w_{uv}, \quad \text{---------- 3.4}$$

where δ $(rvi = r)$ is 1 if $rvi = r$, and 0 otherwise. Once this has been computed for every possible rating value, the predicted rating is simply the value r for which v_{ir} is the greatest.

Example 2: Suppose once again that the two nearest-neighbors of Eric are Lucy and Diane with respective similarity weights 0.75 and 0.15. In this case, ratings 5 and 3 each have one vote. However, since Lucy's vote has a greater weight than Diane's, the predicted rating will be $\hat{r} = 5$. A classification method that considers normalized ratings can also be defined. Let S' be the set of possible normalized values (that may require discretization), the predicted rating is obtained as:

$$\hat{r}_{ui} = h^{-1}\left(\arg\max_{r \in S'} \sum_{v \in \mathcal{N}_i(u)} \delta(h(r_{vi}) = r)\, w_{uv}\right) \quad \text{---------- 3.5}$$

Regression VS Classification

The choice between implementing a neighborhood-based regression or classification method largely depends on the system's **rating scale**. Thus, if the rating scale is continuous, e.g. ratings in the *Jester* joke recommender system can take any value between −10 and 10, then a regression method is more appropriate. On the contrary, if the rating scale has only a few discrete values, e.g. "good" or "bad", or if the values cannot be ordered in an obvious fashion, then a classification method might be preferable.

Furthermore, since **normalization** tends to map ratings to a continuous scale, it may be harder to handle in a classification approach.

Another way to compare these two approaches is by considering the situation where all neighbors have the same **similarity weight**. As the number of neighbors used in the prediction increases, the rating r_{ui} predicted by the regression approach will tend toward the mean rating of item *i*. Suppose item *i* has only ratings at either end of the rating range, i.e. it is either loved or hated, then the regression approach will make the safe decision that the item's worth is average.

On the other hand, the classification approach will predict the rating as the most frequent one given to *i*. This is more risky as the item will be labeled as either "good" or "bad". However, as mentioned before, taking risk may be be desirable if it leads to serendipitous recommendations.

3.5.3 Item-based Recommendation

While user-based methods rely on the opinion of like-minded users to predict a rating, item-based approaches look at ratings given to similar items.

Example 3: Instead of consulting with his peers, Eric instead determines whether the movie "Titanic" is right for him by considering the movies that he has already seen. He notices that people that have rated this movie have given similar ratings to the movies "Forrest Gump" and "Wall-E". Since Eric liked these two movies he concludes that he will also like the movie "Titanic". This idea can be formalized as follows.

Denote by *Nu*(*i*) the items rated by user *u* most similar to item *i*. The predicted rating of *u* for *i* is obtained as a weighted average of the ratings given by *u* to the items of *Nu*(*i*):

$$\hat{r}_{ui} = \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} r_{uj}}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \qquad \text{---------- 3.6}$$

Example 4: Suppose our prediction is again made using two nearest-neighbors, and that the items most similar to "Titanic" are "Forrest Gump" and "Wall-E", with respective similarity weights 0.85 and 0.75. Since ratings of 5 and 4 were given by Eric to these two movies, the predicted rating is computed as

$$\hat{r} = \frac{0.85 \times 5 + 0.75 \times 4}{0.85 + 0.75} \simeq 4.53.$$

---------- 3.7

Again, the differences in the users' individual rating scales can be considered by normalizing ratings with a h:

$$\hat{r}_{ui} = h^{-1}\left(\frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} h(r_{uj})}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}\right).$$

---------- 3.8

3.5.4 User-based VS Item-based Recommendation – 5 criteria

When choosing between the implementation of a user-based and an item-based neighborhood recommender system, five criteria should be considered:

1. **Accuracy**: The accuracy of neighborhood recommendation methods depends mostly on the ratio between the number of users and items in the system. The similarity between two users in userbased methods, which determines the neighbors of a user, is normally obtained by comparing the ratings made by these users on the same items.

Consider a system that has 10,000 ratings made by 1,000 users on 100 items, and suppose, for the purpose of this analysis, that the ratings are distributed uniformly over the items1. Following Table 4.1, the average number of users available as potential neighbors is roughly 650. However, the average number of common ratings used to compute the similarities is only 1. On the other hand, an item-based method usually computes the similarity between two items by comparing ratings made by the same user on these items. Assuming once more a uniform distribution of ratings, we find an average number of potential neighbors of 99 and an average number of ratings used to compute the similarities of 10.

In general, a small number of high-confidence neighbors is by far preferable to a large number of neighbors for which the similarity weights are not trustworthy. In cases where the number of users is much greater than the number of items, such as large commercial systems like *Amazon.com*, item-based methods can therefore produce more accurate recommendations. Likewise, systems that have fewer users than items, e.g., a research paper recommender with thousands of users but hundreds of thousands of articles to recommend, may benefit more from user-based neighborhood methods

Table 1: The average number of neighbors and average number of ratings used in the computation of similarities for user-based and item-based neighborhood methods. A uniform distribution of ratings is assumed with average number of ratings per user $p = |R|/|U|$, and average number of ratings per item $q = |R|/|I|$

	Avg. neighbors	Avg. ratings
User-based	$(\lvert\mathcal{U}\rvert - 1)\left(1 - \left(\frac{\lvert\mathcal{I}\rvert - p}{\lvert\mathcal{I}\rvert}\right)^p\right)$	$\frac{p^2}{\lvert\mathcal{I}\rvert}$
Item-based	$(\lvert\mathcal{I}\rvert - 1)\left(1 - \left(\frac{\lvert\mathcal{U}\rvert - q}{\lvert\mathcal{U}\rvert}\right)^q\right)$	$\frac{q^2}{\lvert\mathcal{U}\rvert}$

2. **Efficiency**: As shown in Table 2, the memory and computational efficiency of recommender systems also depends on the ratio between the number of users and items. Thus, when the number of users exceeds the number of items, as is it most often the case, item-based recommendation approaches require much less memory and time to compute the similarity weights (training phase) than userbased ones, making them more scalable. However, the time complexity of the online recommendation phase, which depends only on the number of available items and the maximum number of neighbors, is the same for user-based and item-based methods.

   Table 2: The space and time complexity of user-based and item-based neighborhood methods, as a function of the maximum number of ratings per user $p = \max_u |I_u|$, the maximum number of ratings per item $q = \max_i |U_i|$, and the maximum number of neighbors used in the rating predictions k. of available items is constantly changing, e.g., an online article recommender, user-based methods could prove to be more stable.

	Space	Time	
		Training	Online
User-based	$O(\|\mathcal{U}\|^2)$	$O(\|\mathcal{U}\|^2 p)$	$O(\|\mathcal{I}\|k)$
Item-based	$O(\|\mathcal{I}\|^2)$	$O(\|\mathcal{I}\|^2 q)$	$O(\|\mathcal{I}\|k)$

3. **Stability:** The choice between a user-based and an item-based approach also depends on the frequency and amount of change in the users and items of the system. If the list of available items is fairly static in comparison to the users of the system, an item-based method may be preferable since the item similarity weights could then be computed at infrequent time intervals while still being able to recommend items to new users. On the contrary, in applications where the list

4. **Justifiability:** An advantage of item-based methods is that they can easily be used to justify a recommendation. Hence, the list of neighbor items used in the prediction, as well as their similarity weights, can be presented to the user as an explanation of the recommendation. By modifying the list of neighbors and/or their weights, it then becomes possible for the user to participate interactively in the recommendation process. User-based methods, however, are less amenable to this process because the active user does not know the other users serving as neighbors in the recommendation.

5. **Serendipity:** In item-based methods, the rating predicted for an item is based on the ratings given to similar items. Consequently, recommender systems using this approach will tend to recommend to a user items that are related to those usually appreciated by this user. For instance, in a movie recommendation application, movies having the same genre, actors or director as those highly rated by the user are likely to be recommended. While this may lead to safe recommendations, it does less to help the user discover different types of items that he might like as much.

   Because they work with user similarity, on the other hand, user-based approaches are more likely to make serendipitous recommendations. This is particularly true if the recommendation is made with a small number of nearestneighbors. For example, a user *A*

that has watched only comedies may be very similar to a user *B* only by the ratings made on such movies. However, if *B* is fond of a movie in a different genre, this movie may be recommended to *A* through his similarity with *B*.

3.6 Components of Neighborhood Methods

Three very important considerations in the implementation of a neighborhood-based recommender system are 1) the normalization of ratings, 2)the computation of the similarity weights, and 3) the selection of neighbors.

3.6.1 Rating Normalization

When it comes to assigning a rating to an item, each user has its own personal scale. Even if an explicit definition of each of the possible ratings is supplied (e.g., 1="strongly disagree", 2="disagree", 3="neutral", etc.), some users might be reluctant to give high/low scores to items they liked/disliked. Two of the most popular rating normalization schemes that have been proposed to convert individual ratings to a more universal scale are *mean-centering* and *Z-score*.

- **Mean-centering**

The idea of mean-centering is to determine whether a rating is positive or negative by comparing it to the mean rating. In user-based recommendation, a raw rating r_{ui} is transformation to a mean-centered one $h(r_{ui})$ by subtracting to r_{ui} the average r_u of the ratings given by user u to the items in I_u:

$$h(r_{ui}) = r_{ui} - \bar{r}_u. \quad \text{---------- 3.9}$$

Using this approach the user-based prediction of a rating *rui* is obtained as

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v)}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}.$$

-- ---------- 3.10

In the same way, the *item*-mean-centered normalization of r_{ui} is given by

$$h(r_{ui}) = r_{ui} - \bar{r}_i,$$

where $\bar{r}_i$ corresponds to the mean rating given to item *i* by user in *Ui*. This normalization technique is most often used in item-based recommendation, where a rating r_{ui} is predicted as:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} (r_{uj} - \bar{r}_j)}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \quad \text{---------- 3.11}$$

An interesting property of mean-centering is that one can see right-away if the appreciation of a user for an item is positive or negative by looking at the sign of the normalized rating. Moreover, the module of this rating gives the level at which the user likes or dislikes the item.

Example 5. As shown in Figure 2, although Diane gave an average rating of 3 to the movies "Titanic" and "Forrest Gump", the user-mean-centered ratings show that her appreciation of these movies is in fact negative. This is because her ratings are high on average, and so, an average rating correspond to a low degree of appreciation. Differences are also visible while comparing the two types of mean-centering. For instance, the item-mean-centered rating of the movie "Titanic" is neutral, instead of negative, due to the fact that much lower ratings were given to that movie. Likewise, Diane's appreciation for "The Matrix" and John's distaste for "Forrest Gump" are more pronounced in the item-mean-centered ratings.

User mean-centering:

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	2.50	-1.50		-0.50	-0.50
Lucy	-2.60	1.40	-1.60	1.40	1.40
Eric	-1.50		-0.50	1.50	0.50
Diane	0.25	-0.75	1.25	-0.75	

Item mean-centering:

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	2.00	-2.00		-1.75	-1.67
Lucy	-2.00	2.00	-1.33	1.25	1.33
Eric	-1.00		-0.33	1.25	0.33
Diane	1.00	0.00	1.67	-0.75	

Figure 2: The *user* and *item* mean-centered ratings of Figure 1

❖ **Z-score normalization**

Consider, two users A and B that both have an average rating of 3. Moreover, suppose that the ratings of A alternate between 1 and 5, while those of B are always 3.

A rating of 5 given to an item by B is more exceptional than the same rating given by A, and, thus, reflects a greater appreciation for this item.

While mean-centering removes the offsets caused by the different perceptions of an average rating, Zscore normalization also considers the spread in the individual rating scales.

Once again, this is usually done differently in user-based than in item-based recommendation. In user-based methods, the normalization of a rating r_{ui} divides the *user*-mean-centered rating by the standard deviation σu of the ratings given by user u:

$$h(r_{ui}) = \frac{r_{ui} - \overline{r}_u}{\sigma_u}. \quad \text{---------- 3.12}$$

A user-based prediction of rating r_{ui} using this normalization approach would therefore be

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v)/\sigma_v}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}.$$

obtained as ---------- 3.13

Likewise, the *z*-score normalization of r_{ui} in item-based methods divides the *item* mean-centered rating by the standard deviation of ratings given to item *i*:

$$h(r_{ui}) = \frac{r_{ui} - \bar{r}_i}{\sigma_i}.$$

---------- 3.14

The item-based prediction of rating *rui* would then be

$$\hat{r}_{ui} = \bar{r}_i + \sigma_i \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} (r_{uj} - \bar{r}_j)/\sigma_j}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}.$$

---------- 3.15

Choosing a Normalization scheme -Comparing mean-centering with Z-score

1) Z-score has additional benefit of considering the variance in the ratings of individual users or items. This is particularly useful if the rating scale has a wide range of discrete values or if it is continuous.
2) *Z*-score can be more sensitive than mean-centering, because the ratings are divided and multiplied by possibly very different standard deviation values, and predict ratings that are outside the rating scale.
3) A more recent investigation showed Z-score has more significant benefits.

Preference-based filtering - if rating normalization is not possible or does not improve the results, another possible approach to remove the problems caused by the individual rating scale is *preference-based filtering*. The particularity of this approach is that it focuses on predicting the relative preferences of users instead of absolute rating values. Since the rating scale does not change the preference order for items, predicting relative preferences removes the need to normalize the ratings.

3.6.2 Similarity Weight Computation

The similarity weights play a double role in neighborhood-based recommendation methods:

1) they allow the selection of trusted neighbors whose ratings are used in the prediction

2) they provide the means to give more or less importance to these neighbors in the prediction.

- ***Correlation-based similarity***

  A measure of the similarity between two objects a and b, often used in information retrieval, consists in representing these objects in the form of two vectors x_a and x_b and computing the Cosine Vector (CV) (or Vector Space) similarity between these vectors:

$$\cos(\mathbf{x}_a, \mathbf{x}_b) = \frac{\mathbf{x}_a^\top \mathbf{x}_b}{||\mathbf{x}_a|| ||\mathbf{x}_b||}. \quad \text{---------- 3.16}$$

In the context of item recommendation, this measure can be employed to compute user similarities by considering a user u as a vector $\mathrm{x}_u \in \mathrm{R}^{|I|}$, where $\mathrm{x}_{ui} = r_{ui}$ if user u has rated item i, and 0 otherwise. The similarity between two users u and v would then be computed as

$$CV(u,v) = \cos(\mathbf{x}_u, \mathbf{x}_v) = \frac{\sum_{i \in \mathcal{I}_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_u} r_{ui}^2 \sum_{j \in \mathcal{I}_v} r_{vj}^2}}, \quad \text{---------- 3.17}$$

where I_{uv} once more denotes the items rated by both u and v. A problem with this measure is that is does not consider the differences in the mean and variance of the ratings made by users u and v. A popular measure that compares ratings where the effects of mean and variance have been removed is the *Pearson Correlation* (PC) similarity:

$$\mathrm{PC}(u,v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \bar{r}_v)^2}}. \quad \text{---------- 3.18}$$

The same idea can be used to obtain similarities between two items i and j, this time by comparing the ratings made by users that have rated both of these items:

$$\mathrm{PC}(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_j)^2}}. \quad \text{---------- 3.19}$$

While the sign of a similarity weight indicates whether the correlation is direct or inverse, its magnitude (ranging from 0 to 1) represents the strength of the correlation.

Example 6. The similarities between the pairs of users and items of our toy example, as computed using PC similarity, are shown in Figure 3. We can see that Lucy's taste in movies is very close to Eric's (similarity of 0.922) but very different from John's (similarity of −0.938). This means that Eric's ratings can be trusted to predict Lucy's, and that Lucy should discard John's opinion on movies or consider the opposite. We also find that the people that like "The Matrix" also like "Die Hard" but hate "Wall-E". Note that these relations were discovered without having any knowledge of the genre, director or actors of these movies.

User-based Pearson correlation

	John	Lucy	Eric	Diane
John	1.000	-0.938	-0.839	0.659
Lucy	-0.938	1.000	0.922	-0.787
Eric	-0.839	0.922	1.000	-0.659
Diane	0.659	-0.787	-0.659	1.000

Item-based Pearson correlation

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
Matrix	1.000	-0.943	0.882	-0.974	-0.977
Titanic	-0.943	1.000	-0.625	0.931	0.994
Die Hard	0.882	-0.625	1.000	-0.804	-1.000
Forrest Gump	-0.974	0.931	-0.804	1.000	0.930
Wall-E	-0.977	0.994	-1.000	0.930	1.000

Figure 3: The *user* and *item* PC similarity for the ratings of Figure 1

The differences in the rating scales of individual users are often more pronounced than the differences in ratings given to individual items. Therefore, while computing the item similarities, it may be more appropriate to compare ratings that are centered on their *user* mean, instead of their *item* mean. The *Adjusted Cosine* (AC) similarity, is a modification of the PC item similarity which compares user-mean-centered ratings:

$$AC(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_u)(r_{uj} - \overline{r}_u)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_u)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \overline{r}_u)^2}}. \quad \text{---------- 3.20}$$

❖ ***Mean Squared Difference*** **(MSD)**

MSD evaluates the similarity between two users u and v as the inverse of the average squared difference between the ratings given by u and v on the same items:

$$\text{MSD}(u,v) = \frac{|\mathcal{I}_{uv}|}{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - r_{vi})^2}. \quad \text{---------- 3.21}$$

While it could be modified to compute the differences on normalized ratings, the MSD similarity is limited compared to PC similarity because it does not capture negative correlations between user preferences or the appreciation of different items. Having such negative correlations may improve the rating prediction accuracy

❖ ***Spearman Rank Correlation*** **(SRC)**

While PC uses the rating values directly, SRC instead considers the ranking of these ratings. Denote by k_{ui} the rating rank of item i in user u's list of rated items (tied ratings get the average rank of their spot). The SRC similarity between two users u and v is evaluated as:

$$\mathrm{SRC}(u,v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (k_{ui} - \bar{k}_u)(k_{vi} - \bar{k}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (k_{ui} - \bar{k}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (k_{vi} - \bar{k}_v)^2}}, \quad \text{---------- 3.22}$$

where $\bar{k}_u$ is the average rank of items rated by u (which can differ from $|I_u|+1$ if there are tied ratings). The principal advantage of SRC is that it avoids the problem of rating normalization, described in the last section, by using rankings. On the other hand, this measure may not be the best one when the rating range has only a few possible values, since that would create a large number of tied ratings. Moreover, this measure is typically more expensive than PC as ratings need to be sorted in order to compute their rank.

Table 3 shows the user-based prediction accuracy (MAE) obtained with MSD, SRC and PC similarity measures, on the *MovieLens*[2] dataset. Results are given for different values of k, which represents the maximum number of neighbors used in the predictions. For this data, we notice that MSD leads to the least accurate predictions, possibly due to the fact that it does not take into account negative correlations. Also, these results show PC to be slightly more accurate than SRC.

Table 3: The rating prediction accuracy (MAE) obtained using the Mean Squared Difference (MSD), the Spearman Rank Correlation and the Pearson Correaltion (PC) similarity. Results are shown for predictions using an increasing number of neighbors k.

k	MSD	SRC	PC
5	0.7898	0.7855	0.7829
10	0.7718	0.7636	0.7618
20	0.7634	0.7558	0.7545
60	0.7602	0.7529	0.7518
80	0.7605	0.7531	0.7523
100	0.7610	0.7533	0.7528

3.6.3 Neighborhood Selection

The number of nearest-neighbors to select and the criteria used for this selection can also have a serious impact on the quality of the recommender system. The selection of the neighbors used in the recommendation of items is normally done in two steps:

1) a global filtering step where only the most likely candidates are kept, and

2) a per prediction step which chooses the best candidates for this prediction.

- ***Pre-filtering of neighbors***

In large recommender systems that can have millions of users and items, it is usually not possible to store the (non-zero) similarities between each pair of users or items, due to memory limitations. Moreover, doing so would be extremely wasteful as only the most significant of these values are used in the predictions. The pre-filtering of neighbors is an essential step that makes neighborhood-based approaches practicable by reducing the amount of similarity

weights to store, and limiting the number of candidate neighbors to consider in the predictions. There are several ways in which this can be accomplished:

- **Top-*N* filtering:** For each user or item, only a list of the N nearest-neighbors and their respective similarity weight is kept. To avoid problems with efficiency or accuracy, N should be chosen carefully. Thus, if N is too large, an excessive amount of memory will be required to store the neighborhood lists and predicting ratings will be slow. On the other hand, selecting a too small value for N may reduce the coverage of the recommendation method, which causes some items to be never recommended.
- **Threshold filtering:** Instead of keeping a fixed number of nearest-neighbors, this approach keeps all the neighbors whose similarity weight has a magnitude greater than a given threshold $w_{\min}$. While this is more flexible than the previous filtering technique, as only the most significant neighbors are kept, the right value of $w_{\min}$ may be difficult to determine.
- **Negative filtering:** In general, negative rating correlations are less reliable than positive ones. Intuitively, this is because strong positive correlation between two users is a good indicator of their belonging to a common group (e.g., teenagers, science-fiction fans, etc.). However, although negative correlation may indicate membership to different groups, it does not tell how different these groups are, or whether these groups are compatible for other categories of items. While experimental investigations have found negative correlations to provide no significant improvement in the prediction accuracy, whether such correlations can be discarded depends on the data.

❖ ***Neighbors in the predictions***

Once a list of candidate neighbors has been computed for each user or item, the prediction of new ratings is normally made with the k-nearest-neighbors, that is, the k neighbors whose similarity weight has the greatest magnitude.

Which value to use for k?

As shown in Table 3, the prediction accuracy observed for increasing values of k typically follows a concave function. Thus, when the number of neighbors is restricted by using a small k (e.g., $k < 20$), the prediction accuracy is normally low. As k increases, more neighbors contribute to the prediction and the variance introduced by individual neighbors is averaged out. As a result, the prediction accuracy improves.

Finally, the accuracy usually drops when too many neighbors are used in the prediction (e.g., $k > 50$), due to the fact that the few strong local relations are “diluted” by the many weak ones. Although a number of neighbors between 20 to 50 is most often described in the literature, see, the optimal value of k should be determined by cross-validation.

More serendipitous recommendations may be obtained at the cost of a decrease in accuracy, by basing these recommendations on a few very similar users. For example, the system could find the user most similar to the active one and recommend the new item that has received the highest rated from this user.

ISBN: 978-81-989720-3-3

UNIT IV ATTACK-RESISTANT RECOMMENDER SYSTEMS

Introduction - Types of Attacks - Detecting attacks on recommender systems - Individual attack - Group attack - Strategies for robust recommender design - Robust recommendation algorithms.

4.1 INTRODUCTION

The input to recommender systems is typically provided through open platforms. Almost anyone can register and submit a review at sites such as Amazon.com and Epinions.com. Like any other data-mining system, the effectiveness of a recommender system depends almost exclusively on the quality of the data available to it. Unfortunately, there are significant motivations for participants to submit incorrect feedback about items for personal gain or for malicious reasons:

• The manufacturer of an item or the author of a book might submit fake (positive) reviews on Amazon in order to maximize sales. Such attacks are also referred to as ***product push*** attacks.

• The competitor of an item manufacturer might submit malicious reviews about the item. Such attacks are also referred to as ***nuke*** attacks.

By creating a concerted set of fake feedbacks from many different users, it is possible to change the predictions of the recommender system. Such users become ***shills*** in the attack process. Therefore, such attacks are also referred to as ***shilling attacks***.

The person making the attack on the recommender system is referred to as the ***adversary***. An adversary would need to create a large number of fake users (or fake *profiles*) to achieve the desired outcome. The number of injected profiles may depend on the specific recommendation algorithm being attacked, and the specific approach used to attack it.

- An attack that requires a smaller number of injected profiles is referred to as an ***efficient* attack** because such attacks are often difficult to detect.
- If an attack requires a large number of injected profiles, then such an attack is ***inefficient*** because most systems should be able to detect a sudden injection of a large number of ratings about a small number of items.
- **Low knowledge attacks** – Require only limited knowledge about the ratings distribution.
- **High knowledge attacks** – Require large knowledge about the ratings distribution.

  A trade-off exists between the amount of knowledge required to make an attack and the efficiency of the attack. If adversaries have more knowledge about the ratings distribution, then they can generally make more efficient attacks.

Understanding the Trade-Offs in Attack Models

Attack models have a number of natural trade-offs between the efficiency of the attack, the amount of knowledge and the specific recommendation algorithm being used required to mount a successful attack.

ISBN: 978-81-989720-3-3

A naive attack: injecting fake user profiles with a single pushed item

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	?	?	7	?	?
Fake-2	?	?	6	?	?
Fake-3	?	?	7	?	?
Fake-4	?	?	6	?	?
Fake-5	?	?	7	?	?

Slightly better than naive attack: injecting fake user profiles with a single pushed item and random ratings on other items

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	2	4	7	6	1
Fake-2	7	2	6	1	5
Fake-3	2	1	7	6	7
Fake-4	1	7	6	2	4
Fake-5	3	5	7	7	4

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	6	7	7	2	1
Fake-2	7	7	6	1	1
Fake-3	1	1	7	6	7
Fake-4	1	1	6	7	6
Fake-5	2	1	7	7	7

Highly knowledgeable attacker injects fake user profiles

Algorithms that are less prone to attacks are referred to as *robust* algorithms. It is one of the goals of recommender systems to design algorithms that are more robust to attacks.

The aforementioned examples lead us to the following observations:

1. Carefully designed attacks are able to affect the predictions with a small number of fake profile insertions. On the other hand, a carelessly injected attack may have no effect on the predicted ratings at all.

2. When more knowledge about the statistics of the ratings database is available, an attacker is able to make more efficient attacks. However, it is often difficult to obtain a significant amount of knowledge about the ratings database.

3. The effectiveness of an attack algorithm depends on the specific algorithm being attacked.

4.2 Types of Attacks

In recommendation systems, attackers may manipulate ratings to either **push** (increase the rating of a target item) or **nuke** (decrease the rating of a target item) specific items. However, to make these attacks more effective, the attacker typically doesn't just manipulate a single item but also injects ratings for other items, called **filler items**, to make the fake profiles more convincing.

1. Filler Items

- **Filler Items**: These are additional items rated by the fake profiles besides the target item. Including filler items helps in making the fake profiles appear more like legitimate user profiles, avoiding detection by automated systems.
- **Null Items**: These are items that the fake profile does not rate. Similar to genuine user profiles, where users only rate a subset of available items, null items are not specified in fake profiles.
- **Effectiveness of Filler Items**: Filler items are particularly important when the target item has a correlation with other items in terms of user ratings. For example:
    - If a target movie like *Gladiator* is often rated together with *Nero*, adding ratings for *Nero* can make a push or nuke attack on *Gladiator* more effective.
    - Adding ratings for an unrelated movie, like *Shrek*, would not be as impactful because it doesn't have the same rating pattern correlation.

2. Trade-Off in Attack Efficiency

- **Knowledge Requirements**: Identifying correlated filler items requires knowledge of the ratings distribution. Therefore, there is a trade-off between how efficient the attack can be and how much information the attacker must gather about the system's rating patterns.
- **Efficiency**: More targeted attacks (using correlated filler items) are generally more effective but require more effort to identify the relationships between items.

3. Types of Attacks

- **Push Attacks**: The goal is to artificially increase the rating of a target item, causing it to appear more popular or highly rated than it should be. This attack is often more efficient when the ratings of items correlated with the target item are manipulated as well (using filler items).
- **Nuke Attacks**: This type of attack aims to reduce the rating of a target item by adding low ratings to it. Nuke attacks tend to have a stronger effect on **hit-ratio**, which measures the number of recommended items that are relevant, as only a few top items are recommended in many systems.
    - **Effect on Hit-Ratio**: Nuke attacks can drastically reduce the hit-ratio because a few bad ratings for a top item can significantly lower its recommendation priority.

- **Effect on Prediction Shift**: While nuke attacks affect the hit-ratio significantly, the effect on prediction shift (the difference between predicted ratings and actual ratings) may be smaller.

Here are more detailed notes on each of the attack methods described in the text:

4.2.1 Random Attack

- **Concept**: In this attack, the ratings for filler items (items other than the target item) are assigned randomly, based on a probability distribution centered around the global mean of ratings. This ensures that the ratings are somewhat consistent with the general distribution of ratings but are not tied to specific items.
- **Target Item**: The target item is assigned the maximum rating (rmax) in a push attack, or the minimum rating (rmin) in a nuke attack.
- **Knowledge Required**: The attacker only needs to know the global mean of ratings across all items. This makes the attack relatively easy to carry out but not highly efficient.
- **Efficiency**: The attack is not very efficient because the randomness of the filler items makes the injected profile less likely to closely resemble the target user profile, making it easy to detect. However, it is a low-effort attack to mount.

4.2.2 Average Attack

- **Concept**: Similar to the random attack, but the ratings for filler items are more deliberately chosen. The ratings for each filler item are assigned based on the average rating for that item. This is done for each filler item, not just for random assignment.
- **Target Item**: The target item is again assigned either the maximum or minimum rating, depending on whether it is a push or nuke attack.
- **Knowledge Required**: The attacker needs to know the global mean of ratings as well as the average rating for each filler item. This requires more knowledge than the random attack, making it slightly more difficult to execute.
- **Conspicuousness**: This attack is more conspicuous than the random attack because the same set of filler items is used in each injected profile. To reduce detectability, random filler items could be chosen for each profile, but this increases the amount of knowledge required (e.g., the global mean of each filler item).
- **Efficiency**: More efficient than the random attack because the ratings of filler items are more specific to each item, making the fake profile more similar to real profiles.

4.2.3 Bandwagon Attack

- **Concept**: The bandwagon attack takes advantage of the popularity of certain items. Popular items are rated with their maximum possible rating (rmax) in the fake user profile. By using popular items as filler, the fake profile is more likely to resemble the profiles of real users, especially when user-based collaborative filtering algorithms are used.
- **Target Item**: As with other attacks, the target item is assigned the maximum or minimum rating (rmax or rmin).
- **Knowledge Required**: The attacker must know which items are popular or widely liked, but not necessarily by analyzing the ratings matrix. Popular items can be

determined from external sources, like product sales or general popularity in public databases.

- **Efficiency**: The attack is more efficient than the average attack in many cases because it focuses on popular items, which are more likely to be positively rated by real users. However, it has limitations when used with item-based collaborative filtering algorithms, where item similarity is more important.
- **Advantages**: Relatively low knowledge requirements compared to the average attack while still being effective in influencing user-based collaborative filtering systems.

4.2.4 Popular Attack

- **Concept**: The popular attack uses a larger set of popular items compared to the bandwagon attack. However, instead of focusing just on widely liked items, the popular attack also includes items that are disliked but widely rated.
- **Target Item**: The target item is always set to the maximum rating in a push attack, or to the minimum rating in a nuke attack.
- **Knowledge Required**: The attacker needs to know the average ratings of popular items. This means more information about the ratings database is required compared to the bandwagon attack.
- **Efficiency**: This attack is generally more powerful than the bandwagon attack because it requires a larger set of popular items, which allows for a greater manipulation of the recommendation process. However, it is also more complex to execute because of the additional knowledge required.
- **Targeted Outcome**: The ratings for popular items are adjusted to increase the chance of finding fake profiles that are similar to the target user, improving the chances of pushing or nuking the target item effectively.

4.2.5 Love/Hate Attack

- **Concept**: This attack is specifically designed for nuke attacks and involves minimal knowledge. The target item is set to the minimum possible rating (rmin), and all filler items are assigned the maximum rating (rmax). The idea is to make the target item highly unlikely to be recommended, and other items are inflated to overshadow it.
- **Target Item**: The target item receives a minimum rating, while all other filler items get maximum ratings.
- **Knowledge Required**: Very little. The attacker does not need to know much about the rating distribution.
- **Efficiency**: Highly effective for nuke attacks, especially with user-based collaborative filtering, as it makes the target item highly unpopular and less likely to be recommended. It is ineffective for push attacks and with item-based filtering algorithms.

4.2.6 Reverse Bandwagon Attack

- **Concept**: This attack is a variation of the bandwagon attack but focuses on using widely disliked items as filler items. These items have received many low ratings, which the attacker uses to target a specific item with low ratings.
- **Target Item**: The target item is set to the minimum rating (rmin) in a nuke attack.

- **Knowledge Required**: Similar to the bandwagon attack, the attacker needs to know which items are disliked or unpopular, which can often be found externally.
- **Efficiency**: Highly effective for nuke attacks in item-based collaborative filtering systems, as it targets items that are negatively rated. However, it is less effective in user-based collaborative filtering, where other methods like the average attack may work better.

4.2.7 Probe Attack

- **Concept**: In this attack, the attacker probes the recommendation system to learn more about the correlations between items and the predicted ratings for those items. The attacker creates a seed profile, uses the recommendation system to generate predictions, and adjusts the seed profile based on these predictions.
- **Target Item**: The target item is set to rmax or rmin depending on whether it is pushed or nuked.
- **Knowledge Required**: The attacker needs to understand how the recommendation system works and may need to adjust the profile iteratively based on the system's responses.
- **Efficiency**: More efficient because it directly uses the recommendation system to gather realistic information about the target item and relevant filler items. This makes the fake profiles more realistic and harder to detect.

4.2.8 Segment Attack

- **Concept**: In a segment attack, the attacker uses domain knowledge to identify a set of users (a segment) who are likely to have preferences similar to the target item. The attack works by selecting a set of filler items that are similar to the target item, pushing it towards a specific group of users.
- **Target Item**: The target item is assigned the maximum rating (rmax) within the segment.
- **Knowledge Required**: The attacker must know the domain (e.g., genres or categories of items) and identify users who are most likely to appreciate the target item. This requires minimal knowledge of the ratings matrix itself.
- **Efficiency**: Effective for push attacks, especially with item-based collaborative filtering, as it targets users who have similar interests. The likelihood of pushing the target item up in recommendations increases because the filler items are related to the target.

4.2.9 Effect of Base Recommendation Algorithm

- **User-Based vs Item-Based Algorithms**: The choice of recommendation algorithm significantly affects the susceptibility to attacks. User-based algorithms are more vulnerable to attacks because they rely heavily on user similarity. Item-based algorithms are generally more robust to attacks.
- **Model-Based Algorithms**: These algorithms tend to be more resilient to attacks due to their focus on patterns learned from data, and hybrid approaches that combine different methods offer even greater robustness.
- **Implicit Feedback**: For implicit data (like clicks or page views), attackers might simulate user actions to manipulate recommendations, similar to how explicit ratings

are manipulated. An example is the "popular page attack," where fake page views are used to push a target page.

These detailed notes cover the various strategies used in attacks on recommendation systems, the knowledge required to implement them, and the effectiveness of each attack depending on the type of recommendation algorithm in use.

4.3 Detecting Attacks on Recommender Systems

An adversarial relationship exists between attackers and the designers of recommender systems. From the point of view of maintaining a robust recommender system, the best way to thwart attacks is to detect them. Detection allows corrective measures (such as the removal of fake user profiles) to be taken.

Accordingly, the detection of fake user profiles is a pivotal element in the design of a robust recommender system. However, the removal of fake profiles is a mistake-prone process, in which genuine profiles might be removed.

It is important not to make too many mistakes, because the removal of authentic profiles can be counter-productive. On the other hand, the inability to remove fake profiles is also undesirable. This results in a natural trade-off between the precision and recall of fake profile removal.

Attack detection algorithms are often measured in terms of this precision and recall. In fact, one can also use1 the receiver operating characteristic (ROC) curve, which plots the trade-off between the true positive rate (TPR) and the false positive rate (FPR).

An alternative way of evaluating the effectiveness of attack removal is by measuring the impact of profile removal on recommender system accuracy.

For example, one can measure the mean absolute error both before and after the filtering of the profiles.

Almost all attacks use multiple profiles in order to undermine the recommender system. The profiles may be removed either **individually** or as a **group** and the attack detection algorithms may be either **supervised** or **unsupervised**.

1. *Unsupervised attack detection algorithms:* In this case, ad hoc rules are used to detect fake profiles. For example, if a profile (or significant portion of it) is identical to many other profiles, then it is likely that all these profiles have been injected for the purpose of creating an attack. The basic idea in this class of algorithms is to identify the key characteristics of attack profiles that are not similar to genuine profiles.
2. *Supervised attack detection algorithms:* Supervised attack detection algorithms use classification models to detect attacks. Individual user profiles or groups of user profiles are characterized as multidimensional feature vectors. In many cases, these multidimensional feature vectors are derived using the same characteristics that are leveraged for the unsupervised case. For example, the number of profiles to which a given user profile is identical can be used as a feature for that user profile. Multiple features can be extracted corresponding to various characteristics of different types of attacks. A binary classifier can then be trained in which known attack profiles are

labeled as +1, and the remaining profiles are labeled as −1. The trained classifier is used to predict the likelihood that a given profile is genuine.

Attack detection methods are either **individual** profile detection methods or **group** profile detection methods. When detecting individual attack profiles, each user profile is assessed independently to determine whether or not it might be an attack. In the case of group detection, a set of profiles is assessed as a group. Note that both the unsupervised and supervised methods can be applied to either individual or group profile detection.

4.4 Individual Attack Profile Detection

Individual attack-profile detection is also referred to as *single attack-profile detection.* If an unsupervised method for individual attack-profile detection is used, a set of features is extracted from each user profile. The features are such that unusually high or unusually low values are indicative of an attack, depending on the feature at hand. In many cases, these features measure the consistency of a particular profile with other profiles in the system.

Other heuristic functions used along with the features are as follows:

1. *Number of prediction differences (NPD):* For a given user, the *NPD* is defined as the number of prediction changes after removing that user from the system. Generally attack profiles tend to have larger prediction differences than usual, because the attack profiles are designed to manipulate the system predictions in the first place.
2. *Degree of disagreement with other users (DD)*: For the ratings matrix R = $[r_{ij}\ ]m\times n$, let ν_j be the mean rating of item j. Then, the degree to which the user i differs from other users on item j is given by $|r_{ij} - \nu_j\ |$. This value is then averaged over all the $|I_i|$ ratings observed for user i to obtain the degree of disagreement DD(i) of user i:

$$DD(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j|}{|I_i|}$$

Users with a larger degree of disagreement with other users are more likely to be attack profiles. This is because attack profiles tend to be different from the distribution of the other ratings.

3. *Rating deviation from mean agreement (RDMA):* The rating deviation from mean agreement is defined as the average absolute difference in the ratings from the mean rating of an item. The mean rating is biased with the inverse frequency if_j of each item j while computing the mean. The inverse frequency if_j is defined as the inverse of the number of users that have rated item j. Let the biased mean rating of an item j be v^b_j. Let I_i be the set of items rated by user i. Then, the value $RDMA(i)$ for user i is defined as follows:

$$RDMA(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j^b| \cdot if_j}{|I_i|}$$

Larger values of this metric indicate the possibility that the user profile might represent an attack.

4. *Standard deviation in user ratings:* This is the standard deviation in the ratings of a particular user. If μ_i is the average rating of user i, and I_i is the set of items rated by that user, then the standard deviation σ_i is computed as follows:

$$\sigma_i = \frac{\sum_{j \in I_i} (r_{ij} - \mu_i)^2}{|I_i| - 1}$$

5. *Degree of similarity with top-k neighbors (SN):* In many cases, attack profiles are inserted in a coordination fashion, with the result being that the similarity of a user with her closest neighbors is increased. Therefore, if w_{ij} is the similarity between the users i and j, and $N(i)$ is the set of neighbors of user i, then the degree of similarity $SN(i)$ is defined as follows:

$$SN(i) = \frac{\sum_{j \in N(i)} w_{ij}}{|N(i)|}$$

The value of w_{ij} can be computed with any standard user-user similarity metric, such as the Pearson correlation coefficient.

The aforementioned features are useful not only for unsupervised attack detection algorithms, but also for supervised methods. The main difference between supervised and unsupervised methods is that examples of previous attacks are available. In such cases, these features are used to create a multidimensional representation and a classification model is constructed. For a given user profile for which the attack behavior is unknown, these features can be extracted. The classification model, which is built on the training data of example attacks, can be used on these features to assess the likelihood that it is indeed an attack.

In addition to these features, a number of generic and model-specific features were introduced. *Model specific features* are designed to detect a specific type of attack, such as an average attack or segment attack. The generic features are as follows:

1. *Weighted deviation from mean agreement (WDMA):* The *WDMA* metric is similar to the *RDMA* metric, but it places greater weight on the ratings of rare items. Therefore, the square of the inverse frequency is used instead of the inverse frequency in the *WDMA* computation.

$$WDMA(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j| \cdot if_j^2}{|I_i|}$$

2. *Weighted degree of agreement (WDA):* The second variation of the *RDMA* metric uses only the numerator of the *RDMA* metric

$$WDA(i) = \sum_{j \in I_i} |r_{ij} - \nu_j| \cdot if_j$$

3. *Modified degree of similarity:* The modified degree of similarity is computed in a similar way to the degree of similarity. The main difference is that the similarity value w_{ij} is proportionally discounted by the number of users who rate both items i and j. This discounting is based on the intuition that the computed similarity is less reliable when the number of items in common between users i and j is small.

4.5 Group Attack Profile Detection

In these cases, the attack profiles are detected as groups rather than as individuals. The basic principle here is that the attacks are often based on groups of related profiles, which are very similar. Therefore, many of these methods use clustering strategies to detect attacks. Some of these methods perform the detection at recommendation time, whereas others use more conventional preprocessing strategies in which detection is performed *a priori*, with the fake profiles are removed up front.

4.5.1 Preprocessing Methods

1. Clustering Approach for Fake Profile Detection

- **Why Clustering?**
  - Fake profiles are designed to manipulate ratings, resulting in highly similar ratings.
  - These similarities make fake profiles form **tight clusters** that are distinct from authentic user profiles.
- **Key Steps in Clustering Approach**:
  1. **Clustering Profiles**:
     - Use algorithms like **PLSA (Probabilistic Latent Semantic Analysis)** to cluster user profiles.
     - PLSA creates **soft clusters** where each user profile has a probability of belonging to each cluster.
  2. **Converting to Hard Clusters**:
     - Assign each profile to the cluster where it has the highest probability.
  3. **Identifying Fake Profile Clusters**:
     - Calculate the **Mahalanobis radius** for each cluster:
       - Smaller Mahalanobis radius = tighter cluster = likely fake profiles.
     - The tightest cluster is assumed to contain fake users.
- **Advantages**:
  - Effective for detecting **overt attacks** where fake profiles exhibit high similarity.

- **Limitations**:
  - Not suitable for detecting **subtle attacks**, where fake profiles mimic authentic behavior more closely.

2. PCA (Principal Component Analysis) Approach for Fake Profile Detection

- **Why PCA?**
  - Fake users often:
    - Have high covariance with each other due to similar ratings.
    - Show low covariance with normal users.
  - PCA identifies dimensions (users) with unusual covariance patterns, revealing fake profiles.
- **Steps in PCA Approach**:
  1. **Normalize Ratings Matrix**:
     - Transform the ratings matrix to have:
       - Zero mean.
       - Unit standard deviation.
  2. **Compute Covariance Matrix**:
     - Use the **transpose of the ratings matrix** (treating users as dimensions).
  3. **Find Smallest Eigenvector**:
     - Compute eigenvectors of the covariance matrix.
     - The **smallest eigenvector** corresponds to dimensions (users) with the least variance (likely fake).
  4. **Identify Fake Users**:
     - Users with **small coefficients** in the smallest eigenvector are flagged as fake profiles.
- **Enhanced PCA Method**:
  - Instead of using a single eigenvector:
    - Consider the top 3 to 5 smallest eigenvectors.
    - Sum the coefficients across these eigenvectors.
    - Users with the smallest summed contributions are identified as fake profiles.

3. UnRAP Algorithm for Detecting Fake Group Profiles

- UnRAP (Unsupervised Rating Profile Analysis) uses a statistical measure called the **Hv-score** to detect fake profiles.
- Origin: Adapted from **biclustering methods in bioinformatics** (used for gene cluster analysis).

Hv-Score Formula

- **Definition**:
  The Hv-score for a user i is defined as:

$$\text{Hv-score}(i) = \frac{\sum_{j \in I_i} (r_{ij} - \mu_i)(r_{ij} - \nu_j)}{\sqrt{\sum_{j \in I_i} (r_{ij} - \mu_i)^2} \cdot \sqrt{\sum_{j \in I_i} (r_{ij} - \nu_j)^2}}$$

Where:

- r_{ij}: Rating by user i for item j.
- μ_i: Mean rating of user i.
- ν_j: Mean rating for item j.
- γ: Mean rating across all users and items.
- I_i: Set of items rated by user i.

Interpretation:

- **Higher Hv-score**: Indicates greater likelihood of a user being a fake profile.
- **Hv-score close to 1**: Suggests random or neutral rating patterns.

Steps in the UnRAP Algorithm

1. **Compute Hv-scores**:
   - Calculate the Hv-score for each user profile.
2. **Identify Suspicious Users**:
   - Select the **top-10 users** with the largest Hv-scores.
   - These users are considered the most likely fake profiles.
3. **Identify the Target Item**:
   - Among the top-10 users, determine the **target item**:
     - The item with the largest deviation from the mean user rating.
   - The target item serves as a reference for detecting additional fake profiles.
4. **Expand Candidate Set**:
   - Relax the criterion to include more than 10 suspicious user profiles.
   - Use a **sliding-window method** to compute a larger candidate set.
5. **Remove False Positives**:
   - Eliminate users who:
     - Did not rate the target item.

- Rated the target item in the **"wrong" direction** (opposite to expected attack behavior).

Features of the Algorithm

- **Key Insight**:
  - Fake profiles tend to have **self-similar ratings** and deviate significantly from authentic user patterns.
- **Detection Approach**:
  - Relies on the statistical properties of the Hv-score's numerator and denominator to capture these patterns.
- **Challenges**:
  - Initial candidate set may include **false positives**, requiring additional filtering.

4.5.3 **Online Methods for Fake Profile Detection**

- Online methods detect fake profiles **in real time** during the recommendation process.
- Primary goal: Improve the robustness of recommendations by identifying and removing fake profiles dynamically.

Key Concept

- **Scenario**:
  - A user-based neighborhood algorithm is used during recommendation.
  - Fake profiles attempt to **push** (inflate ratings) or **nuke** (deflate ratings) a target item.
- **Clustering Active User's Neighborhood**:
  - Divide the active user's neighborhood into **two clusters** based on rating patterns.
  - Analyze these clusters to detect suspicious behavior.

Steps in the Online Method

1. **Cluster Formation**:
   - Create two clusters from the active user's neighborhood (nearby profiles based on similarity).

2. **Analyze Ratings**:
    - Compare the **average ratings** of the target (active) item in both clusters.
    - If the difference in average ratings between the two clusters is **large**, it indicates a potential attack.
3. **Identify the Attack Cluster**:
    - The cluster with the **smaller variance** of ratings for the target item is assumed to be the **attack cluster**.
    - Fake profiles usually show highly similar (low variance) ratings in their attack strategy.
4. **Remove Attack Profiles**:
    - Remove all user profiles in the detected attack cluster.

Integration with Recommendation Systems

- **Direct Integration**:
    - This method can be embedded into the recommendation process to enhance **attack resistance**.
    - During **neighborhood formation**, fake profiles are detected and excluded in real-time.
- **Incremental Updates**:
    - Fake profiles can be identified and removed **incrementally** as the system operates.

Advantages

1. **Real-Time Detection**:
    - Identifies fake profiles at the time of recommendation, ensuring immediate action.
2. **Enhanced Recommendations**:
    - Provides more robust and accurate recommendations by mitigating attack effects.
3. **Efficiency**:
    - Does not require pre-processing; directly integrated into the recommendation algorithm.

Applications

- **E-Commerce Platforms**:
    - Detecting fake reviews or manipulative user behavior during recommendation generation.
- **Media Services**:
    - Identifying bots or malicious profiles attempting to bias content suggestions.

These **online methods** enable real-time detection of fake profiles while simultaneously improving the robustness of recommendations. Their ability to dynamically adapt to attacks makes them a valuable addition to recommendation systems.

4.6 Strategies for Robust Recommender Design

A variety of strategies are available for building recommenders in a more robust way. These strategies range from the use of better recommender-system design to better algorithmic design.

4.6.1 Preventing Automated Attacks with CAPTCHAs

1. Automated Attacks and Their Challenges

- **Attack Scale**:
    - Significant impact on predicted ratings requires **3–5% of authentic profiles** to be fake.
    - Example: For a system with 1 million users, **50,000 fake profiles** may be needed.
- **Automated Profile Creation**:
    - Due to the impracticality of manually creating thousands of fake profiles, attackers use **automated systems** to interact with the web interface and inject fake profiles.

2. Detecting Automated Attacks with CAPTCHAs

- **What are CAPTCHAs?**
    - **Definition**: CAPTCHA stands for **Completely Automated Public Turing Test to Tell Computers and Humans Apart**.
    - Purpose: To differentiate between human users and automated bots.
- **How CAPTCHAs Work**:
    - Present a challenge (e.g., distorted text, images, or puzzles) that is:
        - Easy for humans to solve.

        - Difficult for machines to interpret.
    - Users must correctly decipher and input the challenge to proceed.
- **Integration with Recommender Systems**:
    - Trigger CAPTCHAs in scenarios such as:
        - A large number of rating submissions from the **same IP address**.
        - Unusual patterns of rapid or repetitive interactions with the system.
    - Limit automated interactions while maintaining usability for genuine users.

3. Example of a CAPTCHA

- **Example Challenge**:
    - Users might be shown distorted alphanumeric text or a "select all images with traffic lights" task.
    - Only users who solve the challenge correctly can proceed to submit ratings.

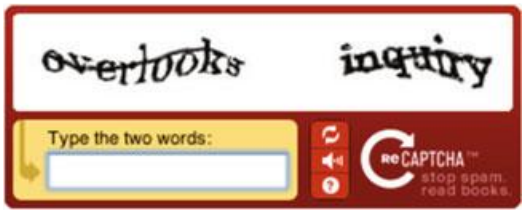


Figure: An example of a CAPTCHA from the official CAPTCHA site (http://www.captcha.net)

4. Advantages of CAPTCHAs

1. **Effective Against Bots**:
    - Prevents automated systems from flooding the system with fake profiles or ratings.
2. **Simple and Inexpensive**:
    - Easy to implement and cost-effective for large-scale systems.
3. **Real-Time Protection**:
    - Can be applied dynamically during profile creation or rating submission.

5. Limitations of CAPTCHAs

1. **User Experience**:
    - Can inconvenience legitimate users, especially if triggered frequently.
2. **Advanced Bots**:
    - Sophisticated bots using AI may bypass simple CAPTCHAs.

3. **Accessibility**:
    - Challenging for users with disabilities (e.g., visually impaired users).

6. Applications

- **E-Commerce and Review Platforms**:
    - Prevent spam reviews and fake ratings.
- **Social Networks**:
    - Stop automated creation of fake user accounts.
- **Online Voting Systems**:
    - Secure against vote manipulation by bots.

CAPTCHAs are a practical tool for detecting and preventing automated attacks, particularly in systems vulnerable to mass fake profile creation. However, balancing security and user convenience is essential for their effective deployment.

4.6.2 Using Social Trust in Recommender Systems

***Social Trust**:*

- Based on relationships and trustworthiness between users.
- Users can specify **trust relationships** based on their experience with others' ratings.
- *Helps to personalize and improve the robustness of recommendations.*

***Effectiveness Against Attacks**:*

- Fake profiles rarely receive trust relationships due to their contrived nature.
- Trust relationships can filter out unreliable profiles, reducing the impact of attacks.

Trust-Based Algorithms

1. **Influence Limiter Algorithm**:
    - Utilizes a **global reputation score** for each user.
    - Reputation scores are incorporated into the recommendation process as weights.
    - Users with higher reputation scores have greater influence on recommendations.

2. **Reputation Calculation**:
    - Reputation is learned by evaluating how accurately a user predicts ratings for their neighbors.
    - A user's reputation reflects their trustworthiness and consistency.
3. **Theoretical Bound on Attack Impact**:
    - The algorithm establishes a limit on how much influence negative attacks can exert on recommendations.

Advantages of Social Trust Methods

1. **Attack Resistance**:
    - By relying on trust, fake profiles are less likely to influence recommendations.
2. **Enhanced Accuracy**:
    - Recommendations are more reliable since they are influenced by trusted users.
3. **Dynamic Trust Learning**:
    - Trust and reputation can evolve over time based on users' behavior.

Challenges and Limitations

1. **Trust Data Availability**:
    - Requires users to specify trust relationships, which may not always be available or accurate.
2. **Computational Complexity**:
    - Calculating and updating reputation scores can be resource-intensive for large datasets.
3. **Trust Propagation Issues**:
    - Trust relationships between users may not always reflect real-world reliability.

Applications of Social Trust in Recommender Systems

1. **E-Commerce**:
    - Identify trusted reviewers or frequent buyers for reliable product recommendations.
2. **Social Networks**:
    - Use friend or follower trust networks to recommend content.
3. **Content Platforms**:
    - Weight user interactions (likes, ratings) by their trustworthiness to suggest better media.

4.6.3 Designing Robust Recommendation Algorithms

It is evident from the discussion in this chapter that different algorithms have different levels of susceptibility to attacks. For example, user-based algorithms are generally much easier to attack than item-based algorithms. Therefore, a number of algorithms have specifically been designed with attack resistance in mind.

4.6.3.1 Incorporating Clustering in Neighborhood Methods

Concept:

User profiles are grouped into clusters using techniques like **PLSA (Probabilistic Latent Semantic Analysis)** or **k-means clustering**.

How It Works:

Each cluster forms an **aggregate profile** by averaging ratings for items within the cluster. Recommendations are made using **closest aggregated profiles** instead of individual user profiles.

Advantages:

- Limits the influence of any single fake profile by blending it into the cluster.
- Provides more robust results compared to traditional nearest-neighbor methods.

Reason for Robustness:

- The clustering process minimizes the impact of outlier profiles by effectively spreading their influence across the cluster.

4.6.3.2 Fake Profile Detection during Recommendation Time

Dual Approach:

- Combines **attack detection** and **recommendation generation**.
- Builds on the method discussed earlier:
    1. **Cluster Division**: The active user's neighborhood is partitioned into two clusters.
    2. **Detecting Attacks**: An attack is suspected if the target item has widely varying average ratings between the two clusters.
    3. **Self-Similarity Check**: The cluster with the smallest radius (most self-similar) is identified as the attack cluster.
    4. **Profile Removal**: Profiles from the attack cluster are excluded from recommendation generation.

Benefits:

- Real-time detection and elimination of attack profiles ensure robust and accurate recommendations.

4.6.3.3 Association-Based Algorithms

- **Rule-Based Collaborative Filtering**:
  - Uses association rules to identify patterns in user-item interactions.
- **Robustness Against Attacks**:
  - **Average Attack**:
    - Resistant to attacks where the maximum fake profile size is less than **15%** of total profiles.
    - Fake profiles typically fail to gain sufficient support to manipulate recommendations.
  - **Segment Attack**:
    - Less effective against segment attacks that target specific user groups or items.
- **Applications**:
  - Suitable for scenarios with strict limits on attack size but may require additional safeguards for segment-level manipulation.

4.6.3.4 Robust Matrix Factorization

Matrix factorization techniques are naturally robust to attacks due to their ability to treat noisy or anomalous data (e.g., attack profiles) as noise during the factorization process. However, specific methods can further enhance their robustness.

1. Natural Robustness of Matrix Factorization

- **Why Matrix Factorization is Robust**:
  - Attack profiles often appear as noise, and the factorization process minimizes their impact by focusing on global patterns in the data.
- **PLSA (Probabilistic Latent Semantic Analysis)**:
  - Many matrix factorization methods are based on PLSA.
  - **Attack Detection Using PLSA**:
    - Attack profiles can be identified during intermediate steps of the algorithm.

- After detection, these profiles are removed, and the **probabilistic parameters** are renormalized for recommendation.

2. Modifying the Optimization Function for Robustness

- **Matrix Factorization Overview**:
  - A ratings matrix R of size m×n (users × items) is factorized into:
    - **User Factors**: U (user preferences) of size m×k
    - **Item Factors**: V (item characteristics) of size n×k
  - The factorization can be expressed as:

$$R \approx UV^T$$

Here $U = [uis]$ and $V = [vjs]$ are $m \times k$ and $n \times k$ matrices. The predicted value $\hat{}rij$ of an entry is as follows:

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} v_{js}$$

Therefore, the error of predicting an observed entry is given by $eij = rij - \hat{}rij$. the matrix entries of U and V are determined by minimizing the sum of squares of eij over all the *observed* entries in the matrix R, along with some regularization terms.

Changing the Objective Function to De-emphasize Attack Profiles

1. Problem with the Standard Objective Function

- In standard matrix factorization, the **Frobenius norm** of the residual matrix is minimized

$$\min \sum_{(i,j) \in S} (R_{ij} - U_i \cdot V_j^T)^2$$

- where S is the set of observed entries in R.
- Attack profiles introduce outliers, creating large residuals |eij| which can disproportionately influence the optimization.

2. Insight for Robustness

- **Outliers and Fake Profiles**:
  - Attack profiles often generate outlier values in the residual matrix $(R - UV^T)$, with large absolute residuals $|e_{ij}| = |R_{ij} - U_i \cdot V_j^T|$.
  - These outliers distort the optimization process, significantly impacting the learned **user factors** U and **item factors** V.

3. Solution: Modifying the Objective Function

To de-emphasize large residuals, alternative loss functions or weighting schemes can be applied:

1. **Huber Loss**:
   - Combines squared error for small residuals and absolute error for large residuals:
   $$L(e_{ij}) = \begin{cases} \frac{1}{2}e_{ij}^2 & \text{if } |e_{ij}| \leq \delta, \\ \delta|e_{ij}| - \frac{1}{2}\delta^2 & \text{if } |e_{ij}| > \delta. \end{cases}$$
   - This approach minimizes the impact of large residuals, making the factorization more robust to attack profiles.

2. **Weighted Loss Function**:
   - Assign lower weights to entries with larger residuals:
   $$\min \sum_{(i,j)\in S} w_{ij}(R_{ij} - U_i \cdot V_j^T)^2$$
   where w_{ij} is a weight inversely proportional to $|e_{ij}|$.

3. **Robust Loss Functions**:
   - Use loss functions that naturally de-emphasize outliers, such as:
     - **Absolute Error**: $|e_{ij}|$
     - **Log-Cosh Loss**: Smooth and less sensitive to outliers:
     $$L(e_{ij}) = \log(\cosh(e_{ij}))$$

4. **Regularization on Outlier Detection**:
   - Introduce a penalty term that explicitly detects and discounts attack profiles.
   - For example, penalize users or items with consistently large residuals.

4. Modified Objective Function

Incorporate these ideas into the optimization problem:

$$\min \sum_{(i,j)\in S} L(e_{ij}) + \lambda(\|U\|^2 + \|V\|^2),$$

where $L(e_{ij})$ is a robust loss function, and λ controls regularization.

5. Benefits of Robust Objectives

- Reduces the influence of attack profiles on the learned factors.

- Preserves the accuracy of recommendations for authentic users.
- Provides a principled way to handle outliers without manual removal of suspected fake profiles.

This approach aligns matrix factorization with robust statistical techniques, making it better suited for attack-prone environments.

Robust Matrix Factorization with Damping Update Steps

Robust matrix factorization is aimed at improving the resilience of collaborative filtering methods against attacks by mitigating the influence of fake profiles or outliers during the optimization process. The approach involves modifying the update steps of traditional matrix factorization to cap large gradient components, especially when errors exceed a threshold (Δ). This helps in limiting the influence of anomalies, which may indicate attack profiles.

1. Update Steps

- The process involves cycling through the observed entries in the ratings matrix in random order.
- The **update steps** are similar to traditional matrix factorization, with one key difference:
    - **Capping Gradient Components**: If the error between the predicted and actual rating exceeds a specified threshold Δ\Delta, the absolute value of the gradient is capped. This prevents large gradient values from significantly altering the user and item factors.
- The goal of this adjustment is to reduce the influence of **large errors**, which may be caused by attack profiles, while still refining the factorization for authentic ratings.

2. Convergence

- The update procedure continues iterating through the observed ratings until **convergence** is reached, which typically means that the error (residuals) falls below a desired threshold.
- Unlike traditional matrix factorization that directly minimizes the error over all observed entries, the **damping approach** ensures that large deviations (outliers) caused by fake profiles are less influential during convergence.

3. Limitations

- **Small Attack Profile Sets**: This method is effective when the number of attack profiles is **small** relative to the authentic profiles. When the number of attack profiles is limited, their influence on the optimization process can be controlled, and the algorithm can still produce reliable recommendations.

- **Large Attack Profile Sets**: If the number of fake profiles is **very large**, this damping approach may not work effectively. Large-scale attacks will distort the matrix factorization process, as the attack profiles will contribute significantly to the residuals, overwhelming the regularization mechanism.

4. Similarity to Robust Regression

- **Robust Matrix Factorization ≈ Robust Regression**: The technique used in robust matrix factorization is conceptually similar to **robust regression**, where the loss function is modified to reduce the impact of outliers in the data.
    - In robust regression, the optimization function is adjusted to minimize the effect of extreme errors, and a similar approach is applied in matrix factorization to deal with anomalies (fake profiles).

5. Potential for Robust Regression in Collaborative Filtering

- While there is no extensive experimental validation for robust regression in collaborative filtering, it is reasonable to assume that **robust regression methods** could improve the resilience of many collaborative filtering models, particularly in situations where there are **outliers** or **attack profiles**.
- This idea presents an interesting avenue for future research in the field, where robust regression could potentially be applied to matrix factorization methods to enhance their attack resistance.

6. Historical Context and Applications

- Robust matrix factorization and **Principal Component Analysis (PCA)** have been widely studied in the context of **recovering the structure of corrupted matrices**. Techniques for identifying and mitigating the effects of anomalies have rich theoretical foundations in areas like **robust statistics** and **data recovery**.
- These approaches can help in improving recommendation systems' ability to handle not only attacks but also noisy or incomplete data.

Summary

- The robust matrix factorization approach modifies the traditional matrix factorization procedure by dampening the impact of large errors, potentially caused by attack profiles.
- This technique works well when the number of fake profiles is small but struggles when the attacks are large-scale.
- It is conceptually similar to robust regression and can be an effective way to create more **attack-resistant collaborative filtering models**. Future research may explore combining **robust regression** with collaborative filtering to improve robustness further.

ISBN: 978-81-989720-3-3

UNIT V EVALUATING RECOMMENDER SYSTEMS

Evaluating Paradigms - User Studies - Online and Offline evaluation - Goals of evaluation design - Design Issues - Accuracy metrics - Limitations of Evaluation measures

INTRODUCTION

A proper design of the evaluation system is crucial in order to obtain an understanding of the effectiveness of various recommendation algorithms. Recommender systems can be evaluated using either online methods or offline methods. In an online system, the user reactions are measured with respect to the presented recommendations. Therefore, user participation is essential in online systems.

For example, in an online evaluation of a news recommender system, one might measure the conversion rate of users clicking on articles that were recommended. Such testing methods are referred to as A/B testing, and they measure the direct impact of the recommender system on the end user. Increasing the conversion rate on profitable items is the most important goal of a recommender system, and it can provide a true measure of the effectiveness of the system.

Testing over multiple data sets is particularly important for assuring greater generalization power of the recommender system so that one can be assured that the algorithm works under a variety of settings. In such cases, offline evaluations with historical data sets are used. Offline methods are, by far, the most common methods for evaluating recommender systems from a research and practice perspective.

When working with offline methods, accuracy measures can often provide an incomplete picture of the true conversion rate of a recommender system.

Important Issues in Designing evaluation methods for Recommender systems

1. *Evaluation goals:* While it is tempting to use accuracy metrics for evaluating recommender systems, such an approach can often provide an incomplete picture of the user experience. Many secondary goals such as **novelty, trust, coverage, and serendipity** are important to the user experience. This is because these metrics have important short and long-term impacts on the conversion rates. Nevertheless, the actual quantification of some of these factors is often quite subjective, and there are often no hard measures to provide a numerical metric.

2. *Experimental design issues:* Even when accuracy is used as the metric, it is crucial to design the experiments so that the accuracy is not overestimated or underestimated. For example, if the same set of specified ratings is used both for model construction and for accuracy evaluation, then the accuracy will be grossly overestimated. In this context, careful experimental design is important.

3. *Accuracy metrics*: In spite of the importance of other secondary measures, accuracy metrics continue to be the single most important component in the evaluation. Recommender systems can be evaluated either in terms of the prediction accuracy of a rating or the accuracy of ranking the items. Therefore, a number of common metrics such as the **mean absolute error** and **mean squared error** are used frequently. The evaluation of rankings can be performed with the use of various methods, such as utility-based computations, rank-correlation coefficients, and the receiver operating characteristic curve.

5.1 EVALUATION PARADIGMS

There are three primary types of evaluation of recommender systems:

- User studies
- Online evaluation
- Offline evaluation with historical datasets

The main differences between the first two settings lie in how the users are recruited for the studies. Although online evaluations provide useful insights about the true effects of a recommendation algorithm, there are often significant practical impediments in their deployment.

5.2 User Studies

- **Definition**: A method where users are actively recruited to interact with a recommender system by performing specific tasks. This interaction involves tasks like exploring recommendations, rating items, or providing qualitative feedback. The process collects data on user preferences, behaviors, and their responses to recommendations, which are then analyzed to understand and improve the system's effectiveness.
- **Data Collection**: Feedback before/after interaction and interaction data.
- **Purpose**: Infer user preferences (likes/dislikes). Example: Users rate songs or products to judge recommendation quality and algorithm effectiveness.
- **Advantages**:
    - Detailed user interaction data collection.
    - Test various scenarios (e.g., algorithm or UI changes).
- **Disadvantages**:
    - User awareness may bias responses.
    - Recruitment is expensive and labor-intensive.

- Recruitment filters create biases (e.g., participants may not represent the general population).
- Results lack full trust due to voluntary participation and bias in responses.

5.3 Online Evaluations of Recommender Systems

Online evaluations are essential for assessing the performance of recommender systems in real-world environments. These evaluations leverage data from real users interacting with a deployed or commercial system. Unlike laboratory-based evaluations, this approach reduces recruitment bias, as users engage with the system naturally.

Key Features

1. **Real Users:**
   - Users are not recruited artificially; they participate naturally while interacting with the system.
   - This minimizes bias and ensures the evaluation reflects real-world conditions.
2. **Algorithm Comparison:**
   - Online evaluations often involve testing multiple algorithms on subsets of users.
   - Performance is measured using metrics such as conversion rate, which indicates how often users act on recommendations.

Metrics for Evaluation

1. **Conversion Rate:**
   - Measures the frequency with which users select or engage with recommended items.
   - Example: In a news recommender, the conversion rate is the fraction of times users click on a recommended article.
   - Additional factors, such as item cost or profit, can be incorporated to reflect the importance of the recommendations.
2. **A/B Testing:**
   - A widely used method to compare two algorithms. The steps are as follows:
     1. Users are randomly divided into two groups: Group A and Group B.

2. Group A interacts with one algorithm, while Group B uses another.
3. All other conditions, such as user selection and context, are kept as similar as possible.
4. Metrics (e.g., conversion rates) are compared to determine which algorithm performs better.

- This approach mirrors the methodology of clinical trials in medicine.

Advanced Methods: Multi-Arm Bandit Algorithms

- Multi-arm bandit algorithms are inspired by gambling, where a player (the recommender system) chooses between multiple slot machines (algorithms).
- The goal is to maximize payoff (e.g., user engagement or profit) by balancing:
  1. Exploration: Testing algorithms randomly to gather data on performance (e.g., 10% of trials).
  2. Exploitation: Using the best-known algorithm for most interactions (e.g., 90% of trials).
- The approach dynamically updates based on real-time user feedback.
- Recent performance is often weighted more heavily than older results to improve responsiveness.

Relation to Reinforcement Learning

- Multi-arm bandit algorithms align closely with reinforcement learning (RL), which focuses on balancing short-term exploration with long-term optimization.
- RL has been extensively studied in fields like classification and regression but remains underexplored in the domain of recommender systems.
- Significant research opportunities exist for integrating RL into recommendation algorithms to enhance effectiveness and adaptability.

Advantages of Online Evaluations

1. **Realistic Context:**
   - Users interact with the system in their natural environment, ensuring realistic feedback.
2. **Goal-Oriented Evaluation:**

- Metrics like conversion rate directly measure system goals, such as profit or user satisfaction.

3. **Dynamic Adaptability:**
   - Algorithms adapt in real time, improving their performance as more data is collected.

Limitations of Online Evaluations

1. **Large User Base Required:**
   - Online evaluations need significant user participation, making them unsuitable for startups or early-stage systems.
2. **Limited Accessibility:**
   - These evaluations are often confined to proprietary systems owned by commercial entities.
   - General researchers or third-party evaluators cannot easily access or replicate these settings.
3. **Lack of Generalizability:**
   - Results are system-specific and may not extend to other domains or settings.
4. **Uncontrolled User Actions:**
   - User behavior in online settings cannot be fully controlled, limiting the ability to perform controlled stress-testing.

Opportunities for Improvement

1. **Generalizable Frameworks:**
   - Developing evaluation methods that work across diverse domains and datasets.
2. **Enhanced Algorithms:**
   - Combining reinforcement learning with multi-arm bandit strategies to create adaptive, robust recommender systems.
3. **Simulated Testing Environments:**
   - Designing hybrid evaluation systems that combine online methods with controlled offline simulations to address variability in user behavior.

6 Offline Evaluation with Historical Data Sets

Offline testing is a widely used approach for evaluating recommender systems, relying on pre-collected historical data

to assess algorithm performance. This method allows for standardized and repeatable evaluation without requiring access to live user interactions.

Key Features of Offline Testing

1. **Use of Historical Data**:
    - Relies on datasets containing historical user interactions, such as ratings, clicks, or purchases.
    - Examples include the **Netflix Prize dataset**, a benchmark dataset released during a global contest, widely used to compare algorithms.
2. **Temporal Information**:
    - Historical data often includes **timestamps**, indicating when a user interacted with an item.
    - Temporal data helps evaluate how algorithms handle evolving user preferences over time.
3. **Standardized Benchmarks**:
    - Historical datasets from diverse domains (e.g., music, movies, news) are available, enabling comparisons across settings.
    - This allows researchers to test the **generalizability** of recommender systems to different types of data.

Advantages of Offline Testing

1. **No Large User Base Required**:
    - Datasets are pre-collected, eliminating the need for a large live user base.
    - Ideal for academic research or system development before deployment.
2. **Reproducibility**:
    - Using standardized datasets ensures experiments are repeatable and results can be compared across studies.
3. **Statistical Robustness**:
    - Offline testing employs well-established metrics such as **Root Mean Square Error (RMSE)**, **Precision**, **Recall**, and **F1-score**, which provide clear and interpretable results.
4. **Efficient Evaluation**:
    - Algorithms can be tested in a controlled environment, making the process faster and less resource-intensive than real-time evaluations.

Limitations of Offline Testing

1. **Lack of Real-Time Interaction**:
    - Offline methods do not capture the **actual propensity** of a user to interact with recommendations in real life.
    - The absence of real-time feedback means algorithms may not adapt to user behavior effectively.

2. **Evolving Data**:
    - User preferences and data distributions can evolve over time, while offline datasets are static.
    - This limits the ability to evaluate how well algorithms perform in dynamic environments.
3. **Limited Metrics**:
    - Offline metrics often emphasize accuracy but fail to account for critical qualities such as:
        - **Serendipity**: The ability to suggest unexpected but relevant items.
        - **Novelty**: Recommending new and unseen items.
    - These aspects impact long-term goals, like user engagement and conversion rates, which offline methods fail to measure directly.
4. **Lack of Contextual Feedback**:
    - Offline datasets lack information about the context in which users interact with recommendations, such as mood, location, or time of day.

Popular Offline Testing Frameworks

1. **Netflix Prize Dataset**:
    - Released during a contest, it serves as a benchmark for evaluating movie recommendation algorithms.
    - Includes millions of user ratings, timestamps, and user-item interactions.
2. **Diverse Domain Datasets**:
    - Datasets are available for various domains like music (Last.fm dataset), news (MIND dataset), and e-commerce (Amazon review dataset).
    - These help assess the applicability of algorithms across industries.

Reasons for Popularity

1. **Established Frameworks**:
    - Offline testing is supported by a range of well-defined evaluation frameworks, making it accessible and easy to implement.
2. **Clear Metrics**:
    - Metrics like **Mean Absolute Error (MAE)**, **Mean Reciprocal Rank (MRR)**, and **Area Under the Curve (AUC)** provide quantifiable and interpretable results.
3. **Academic and Research Use**:
    - Widely accepted in academia due to ease of use, reproducibility, and availability of datasets.

ISBN: 978-81-989720-3-3

5.4 General Goals of Evaluation Design

The evaluation of recommender systems focuses on various goals beyond the basic measure of accuracy. These goals include **diversity**, **serendipity**, **novelty**, **robustness**, and **scalability**, among others. While some of these goals are objective and quantifiable, others are subjective and depend on user experience, often measured via user surveys.

1. Accuracy

- Accuracy evaluates how close the recommendations are to the true preferences of users.

  Let R be the ratings matrix in which r_{uj} is the known rating of user u for item j. Consider the case where a recommendation algorithm estimates this rating as $\hat{r}_{uj}$. Then, the entry-specific error of the estimation is given by the quantity $e_{uj} = \hat{r}_{uj} - r_{uj}$. The overall error is computed by averaging the entry-specific errors either in terms of absolute values or in terms of squared values. Furthermore, many systems do not predict ratings; rather they only output rankings of top-k recommended items. This is particularly common in implicit feedback data sets. Different methods are used to evaluate the accuracy of ratings predictions and the accuracy of rankings.

  Main components of accuracy evaluation are as follows:

  1. Designing the accuracy evaluation: All the observed entries of a ratings matrix cannot be used both for training the model and for accuracy evaluation. Doing so would grossly overestimate the accuracy because of overfitting. It is important to use only a different set of entries for evaluation than was used for training. If S is the observed entries in the ratings matrix, then a small subset $E \subset S$ is used for evaluation, and the set $S - E$ is used for training. This issue is identical to that encountered in the evaluation of classification algorithms.

  2. Accuracy metrics: Accuracy metrics are used to evaluate either the prediction accuracy of estimating the ratings of specific user-item combinations or the accuracy of the top k ranking predicted by a recommender system. Typically, the ratings of a set E of entries in the ratings matrix are hidden, and the accuracy is evaluated over these hidden entries. Different classes of methods are used for the two cases:

• Accuracy of estimating ratings: As discussed above, the entry-specific error is given by euj = ˆruj − ruj for user u and item j. This error can be leveraged in various ways to compute the overall error over the set E of entries in the ratings matrix on which the evaluation is performed. An example is the mean squared error, which is denoted by MSE:

$$MSE = \frac{\sum_{(u,j)\in E} e_{uj}^2}{|E|}$$

The square-root of the aforementioned quantity is referred to as the root mean squared error, or RMSE.

$$RMSE = \sqrt{\frac{\sum_{(u,j)\in E} e_{uj}^2}{|E|}}$$

Accuracy of estimating rankings: Many recommender systems do not directly estimate ratings; instead, they provide estimates of the underlying ranks. Depending on the nature of the ground-truth, one can use rank-correlation measures, utility-based measures, or the receiver operating characteristic. The latter two methods are designed for unary (implicit feedback) data sets.

Problem with accuracy metrics

Some measures of accuracy are also designed to maximize the profit for the merchant because all items are not equally important from the perspective of the recommendation process. These metrics incorporate item-specific costs into the computation. The main problem with accuracy metrics is that they often do not measure the true effectiveness of a recommender system in real settings. For example, an obvious recommendation might be accurate, but a user might have eventually bought that item anyway. Therefore, such a recommendation might have little usefulness in terms of improving the conversion rate of the system.

2. Coverage

Even when a recommender system is highly accurate, it may often not be able to ever recommend a certain proportion of the items, or it may not be able to ever recommend to a certain proportion of the users. This measure is referred to as coverage.

Recommender systems face challenges because rating matrices are often sparse (i.e., many ratings are missing). For instance, if a rating matrix has only one entry per row and column, most algorithms cannot provide useful recommendations. However, different recommender systems vary in how well they handle this issue.

In practice, systems often achieve complete coverage by using default values for missing ratings. A common default is to use the average rating of an item across users when a specific user's rating is unavailable. Because of this, it's important to balance accuracy (how good the recommendations are) with coverage (how many items can be recommended) during evaluation.

Types of Coverage

1. ***User-space coverage*** measures the fraction of users for which at least k ratings may be predicted. The value of k should be set to the expected size of the recommendation list. When fewer than k ratings can be predicted for a user, it is no longer possible to present a meaningful recommendation list of size k to the user. Such a situation could occur when a user has specified very few ratings in common with other users. Consider a user-based neighborhood algorithm. It is difficult to robustly compute the peers of that user, because of very few mutually specified ratings with other users. Therefore, it is often difficult to make sufficient recommendations for that user.

2. Itemspace coverage measures the fraction of items for which the ratings of at least k users can be predicted. In practice, however, this notion is rarely used, because recommender systems generally provide recommendation lists for users, and they are only rarely used for generating recommended users for items. A different form of item-space coverage evaluation is defined by the notion of catalog coverage, which is specifically suited to recommendation lists.

Imagine a scenario where every entry in the ratings matrix can be predicted by an algorithm, but the same set of top k items is always recommended to every user. Therefore, even though the aforementioned definition of item-space coverage would suggest good performance, the actual coverage across all users is very limited. In other words, the recommendations are not diverse across users, and the catalog of items is not fully covered. Let Tu represent the list of top-k items recommended to user u ∈ {1 . . .m}. The catalog coverage CC is defined as the fraction of items that are recommended to at least one user, the notation n represents the number of items.

$$CC = \frac{|\cup_{u=1}^{m} T_u|}{n}$$

Formulas for Coverage:

1. **User-Space Coverage:**

$$\text{User Coverage} = \frac{\text{Number of users with at least one recommendation}}{\text{Total number of users}}$$

2. **Item-Space Coverage:**

$$\text{Item Coverage} = \frac{\text{Number of items recommended}}{\text{Total number of items in the catalog}}$$

Example:

- If a system has 100 items but only recommends 30 unique items, item coverage is 30%.

3. Confidence and Trust

The process of estimating ratings in recommender systems is not precise and depends on the training data and the algorithm used. This creates uncertainty about prediction accuracy, which is why some systems include confidence intervals to indicate their reliability. Smaller confidence intervals are preferred as they build user trust, provided the predictions are accurate within the specified confidence level.

Comparing confidence intervals requires both systems to use the same confidence level (e.g., 95%). If different levels are used (e.g., 95% vs. 99%), meaningful comparisons are impossible. Confidence reflects the system's belief in its predictions, while trust reflects the user's belief in the system. Trust is influenced by factors like logical explanations, even if the predictions are accurate.

Trust and utility often conflict. For instance, recommending familiar items increases trust but may offer little utility. Conversely, recommending novel items improves utility but may lower trust. These trade-offs are inherent in recommender systems. Trust is typically measured through user surveys conducted during experiments.

Confidence Interval Formula:

For a predicted rating $\hat{r}_{uj}$ with standard deviation σ:

$$\text{Confidence Interval} = \hat{r}_{uj} \pm Z \cdot \frac{\sigma}{\sqrt{n}}$$

- Z-score for the desired confidence level (e.g., 1.96 for 95%).

Trust Measurement:

No strict formula; usually measured surveys with Likert scales.via

4. Novelty

The novelty of a recommender system evaluates the likelihood of a recommender system to give recommendations to the user that they are not aware of, or that they have not seen before.

Recommending unseen items helps users discover new likes and dislikes, offering more value than suggesting items they already know about but haven't rated. For example, recommending a book from a niche genre a user hasn't explored is more insightful than suggesting a bestseller they already know exists. While content-based systems often recommend expected items (e.g., books similar to a user's recent reads), such suggestions may build trust but are less effective in driving new engagement or purchases.

Measuring Novelty:

1. **Online Experimentation:** Users are asked directly if they were aware of an item before it was recommended. For example, in a survey, a user might report whether a suggested movie was entirely new to them.
2. **Offline Estimation:** Novelty can be estimated using time-stamped data when online experimentation isn't feasible:
    - Ratings made after a specific time (t_0) are hidden from the training data.

- Some earlier ratings (before t_0) are also removed.
- The system is trained on the reduced data, and its ability to recommend items is evaluated.
- Correct recommendations for items rated before t_0 are penalized (less novel), while those rated after t_0 are rewarded (more novel).

Additionally, novelty scores may give less credit for recommending popular items since they are more likely to be known by users. For instance, suggesting a blockbuster movie scores lower for novelty than recommending an underrated indie film.

Formulas:

1. **Novelty Score:**

$$\text{Novelty} = \frac{\text{Number of novel recommendations}}{\text{Total recommendations}}$$

Example:

- If 5 out of 10 recommended items are new to a user, novelty is 50%.

5. Serendipity

The word "serendipity" literally means "lucky discovery." Therefore, serendipity is a measure of the level of surprise in successful recommendations. In other words, recommendations need to be unexpected. In contrast, novelty only requires that the user was not aware of the recommendation earlier. Serendipity is a stronger condition than novelty.

Consider the case where a particular user frequently eats at Indian restaurants. The recommendation of a new Pakistani restaurant to that user might be novel if that user has not eaten at that restaurant earlier. However, such a recommendation is not serendipitious, because it is well known that Indian and Pakistani food are almost identical. On the other hand, if the recommender system suggests a new Ethiopian restaurant to the user, then such a recommendation is serendipitious because it is less obvious. Therefore, one way of viewing serendipity is as a departure from "obviousness."

Measuring Serendipity in Recommender systems

1. Online methods: The recommender system collects user feedback both on the usefulness of a recommendation and its obviousness. The fraction of recommendations that are both useful and non-obvious, is used as a measure of the serendipity.

2. Offline methods: One can also use a primitive recommender to generate the information about the obviousness of a recommendation in an automated way. The primitive recommender is typically selected as a content-based recommender, which has a high propensity for recommending obvious items. Then, the fraction of the recommended items in the top-k lists that are correct (i.e., high values of hidden ratings), and are also not recommended by the primitive recommender are determined. This fraction provides a measure of the serendipity.

Measurement:

1. Compare recommendations from a baseline system (e.g., popularity-based) with the advanced system.
2. Assign scores based on surprise and relevance.

Example:

- If a user typically watches action movies but finds an enjoyable documentary recommended, that's serendipitous.

6. Diversity

The notion of diversity implies that the set of proposed recommendations within a single recommended list should be as diverse as possible. For example, consider the case where three movies are recommended to a user in the list of top-3 items. If all three movies are of a particular genre and contain similar actors, then there is little diversity in the recommendations. If the user dislikes the top choice, then there is a good chance that she might dislike all of them. Presenting different types of movies can often increase the chance that the user might select one of them. Note that the diversity is always measured over a set of recommendations, and it is closely related to novelty and serendipity. Ensuring greater diversity can often increase the novelty and serendipity of the recommendations. Furthermore, greater diversity of recommendations can also increase the sales diversity and catalog coverage of the system.

Diversity can be measured in terms of the content-centric similarity between pairs of items. The vector-space representation of each item description is used for the similarity computation. For example, if a set of k items are recommended to the user, then the pairwise similarity is computed between every pair of items in the list.

Formula for Pairwise Diversity:

For a recommendation list L:

$$\text{Diversity} = 1 - \frac{1}{|L| \cdot (|L| - 1)} \sum_{i,j \in L, i \neq j} \text{Similarity}(i, j)$$

- Similarity(i,j): Content-based or collaborative similarity between items i and j.

Example:

- A list of books that includes genres like fiction, history, and self-help is more diverse than one that contains only romance novels.

7. Robustness and Stability

A recommender system is stable and robust when the recommendations are not significantly affected in the presence of attacks such as fake ratings or when the patterns in the data evolve significantly over time. In general, significant profit-driven motivations exist for some users to enter fake ratings. For example, the author or publisher of a book might enter fake positive ratings about a book at Amazon.com, or they might enter fake negative ratings about the books of a rival.

Robustness Formula:

Analyzed via performance degradation under adversarial attacks:

$$\text{Robustness} = \frac{\text{Performance after attack}}{\text{Performance before attack}}$$

Stability Metric:

Measures consistency over time:

$$\text{Stability} = 1 - \text{Normalized Difference in Recommendations}$$

8. Scalability

In recent years, it has become increasingly easy to collect large numbers of ratings and implicit feedback information from various users. In such cases, the sizes of the data sets continue to increase over time. As a result, it has become increasingly essential to design recommender systems that can perform effectively and efficiently in the presence of large amounts of data.

Measures used for determining the scalability of a system:

1. Training time: Most recommender systems require a training phase, which is separate from the testing phase. For example, a neighborhood-based collaborative filtering algorithm might require pre-computation of the peer group of a user, and a matrix factorization system requires the determination of the latent factors. The overall time required to train a model is used as one of the measures. In most cases, the training is done offline. Therefore, as long as the training time is of the order of a few hours, it is quite acceptable in most real settings.

2. Prediction time: Once a model has been trained, it is used to determine the top recommendations for a particular customer. It is crucial for the prediction time to be low, because it determines the latency with which the user receives the responses.

3. Memory requirements: When the ratings matrices are large, it is sometimes a challenge to hold the entire matrix in the main memory. In such cases, it is essential to design the algorithm to minimize memory requirements.When the memory requirements become very high, it is difficult to use the systems in large-scale and practical settings.

5.5 Design Issues in Offline Recommender Evaluation

It is crucial to design recommender systems in such a way that the accuracy is not grossly overestimated or underestimated. For example, one cannot use the same set of specified ratings for both training and evaluation. Doing so would grossly overestimate the accuracy of the underlying algorithm. Therefore, only a part of the data is used for training, and the remainder is often used for testing. The ratings matrix is typically sampled in an entry-wise

fashion. In other words, a subset of the entries are used for training, and the remaining

entries are used for accuracy evaluation.

This approach is similar to that used for testing classification and regression modeling algorithms. The main difference is that classification and regression modeling methods sample rows of the labeled data, rather than sampling the entries. This difference is because the unspecified entries are always restricted to the class variable in classification, whereas any entry of the ratings matrix can be unspecified.

The design of recommender evaluation systems is very similar to that of classifier evaluation systems because of the similarity between the recommendation and classification problems. A common mistake made by analysts in the benchmarking of recommender systems is to use the same

data for parameter tuning and for testing. Such an approach grossly overestimates the accuracy because parameter tuning is a part of training, and the use of test data in the training process leads to overfitting. To guard against this possibility, the data are often divided into three parts:

1. Training data: This part of the data is used to build the training model. For example, in a latent factor model, this part of the data is used to create the latent factors from the ratings matrix. One might even use these data to create multiple models in order to eventually select the model that works best for the data set at hand.

2. Validation data: This part of the data is used for model selection and parameter tuning. For example, the regularization parameters in a latent factor model may be determined by testing the accuracy over the validation data. In the event that multiple models have been built from the training data, the validation data are used to determine the accuracy of each model and select the best one.

3. Testing data: This part of the data is used to test the accuracy of the final (tuned) model. It is important that the testing data are not even looked at during the process of parameter tuning and model selection to prevent overfitting. The testing data are used only once at the very end of the process. Furthermore, if the analyst uses the results on the test data to adjust the model in some way, then the results will be contaminated with knowledge from the testing data.

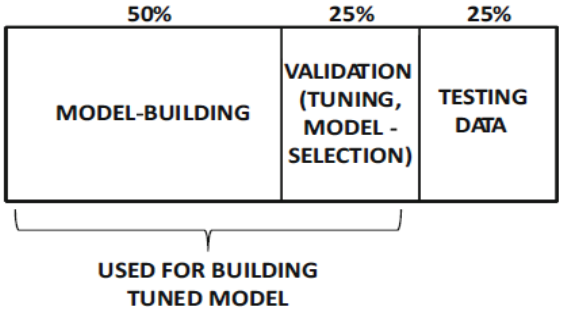
Note that the validation data may also be considered a part of the training data because they are used to create the final tuned model. The division of the ratings matrix into the ratios 2:1:1 is particularly common.

5.5.1 Case Study of the Netflix Prize Data Set

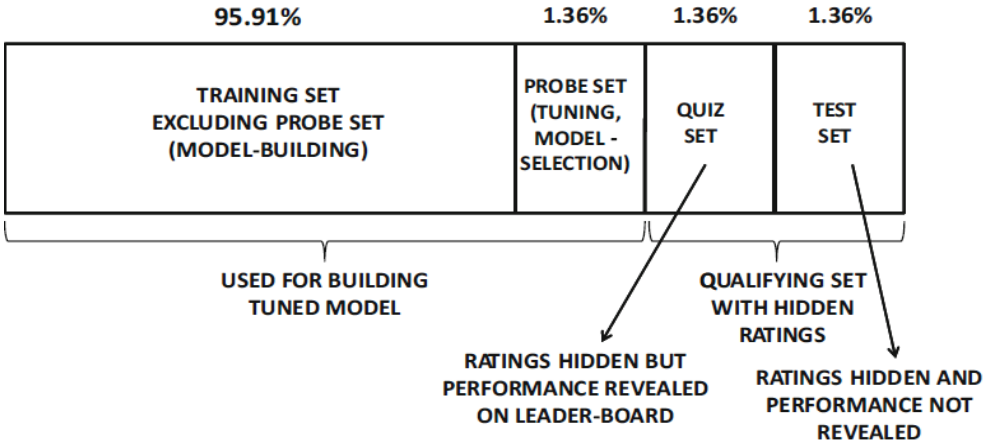
The Netflix Prize dataset is a well-known example used in collaborative filtering, demonstrating how Netflix avoided overfitting by carefully dividing the data. Here's how the dataset was structured:

1. **Training Data (95.91%)**
    - This majority portion was used by participants to build their models.
2. **Probe Set (1.36%)**
    - Served a role similar to a validation set.
    - Used by participants for parameter tuning and model selection.
    - The distribution of ratings in the probe set differed slightly from the training data because it contained more recent ratings, better reflecting the distribution of hidden ratings in the qualifying set.

3. **Qualifying Set (2.7%)**
    - Ratings were hidden; participants were only given triplets of **User, Movie, Date** without the actual ratings.
    - Divided into two parts:
        - **Quiz Set**: Half of the qualifying set, used for leaderboard updates to show participants their progress.
        - **Test Set**: The other half, used for final evaluation.
    - Final prize determination was based only on performance on the test set.



(a) Proportional division of ratings



(b) Division in Netflix Prize data set (not drawn to scale)

- **Avoiding Overfitting:**
  Participants did not know which part of the qualifying set was the quiz set. This ensured that the test set remained a truly "out-of-sample" dataset for unbiased evaluation.

- **Challenge with Quiz Set:**
  While leaderboard feedback helped participants refine their models, repeated submissions risked overfitting to the quiz set.

- **Purpose of Design:**
  The careful division of data ensured fair evaluation by preventing participants from using test set insights during model training.

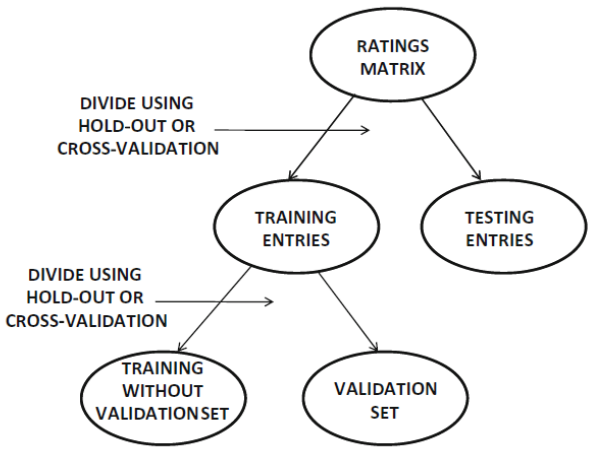
- The Netflix Prize design illustrates the importance of separating test data from the training process to maintain the integrity of model evaluation. This is a gold standard in benchmarking, which is often not strictly followed in practice.
- The overall division of the Netflix data set is shown in Figure 7.1(b). The only difference from the division in Figure 7.1(a) is the presence of an additional quiz set. It is, in fact, possible to remove the quiz set entirely without affecting the Netflix contest in any significant way, except that participants would no longer be able to obtain an idea of the quality of their submissions.
- Indeed, the Netflix Prize evaluation design is an excellent example of the importance of not using any knowledge of the performance on the test set at any stage of the training process until the very end. Benchmarking in research and practice often fails to meet these standards in one form or the other.

5.5.2 Segmenting the Ratings for Training and Testing

In practice, real data sets are not pre-partitioned into training, validation, and test data sets. Therefore, it is important to be able to divide the entries of a ratings matrix into these portions automatically. Most of the available division methods, such as hold-out and crossvalidation, are used to divide1 the data set into two portions instead of three. However, it is possible to obtain three portions as follows. By first dividing the rating entries into training and test portions, and then further segmenting the validation portion from the training data, it is possible to obtain the required three segments.

5.2.2.1 Hold-Out

In the hold-out method, a fraction of the entries in the ratings matrix are hidden, and the remaining entries are used to build the training model. The accuracy of predicting the hidden entries is then reported as the overall accuracy. Such an approach ensures that the reported accuracy is not a result of overfitting to the specific data set, because the entries used for evaluation are hidden during training. Such an approach, however, underestimates the true accuracy. First, all entries are not used in training, and therefore the full power of the data is not used. Second, consider the case where the held-out entries have a higher average rating than the full ratings matrix. This means that the held-in entries have a lower average rating than the ratings matrix, and also the held-out entries. This will lead to a pessimistic bias in the evaluation.



2: Hierarchical division of rated entries into training, validation, and testing

5.2.2.2 Cross-Validation

- The rating entries in the matrix are divided into **q equal parts** (segments).
- Each segment has **|S|/q entries**, where **|S|** is the total number of specified entries in the ratings matrix.

2. **Process:**
   - One of the **q segments** is used as the **test set**, and the remaining **q - 1 segments** are used for **training**.
   - This process is repeated **q times**, with each segment taking a turn as the test set.
   - The **average accuracy** across all **q test sets** is calculated as the final result.
3. **Large q for Accuracy:**
   - When **q** is large, this method closely estimates the true accuracy of the model.
4. **Leave-One-Out Cross-Validation (Special Case):**
   - When **q = |S|**, only one entry is hidden (test set), and the remaining entries are used for training.
   - This is highly accurate but computationally expensive because the model needs to be trained **|S| times**.
5. **Practical Use:**
   - Typically, **q** is set to a smaller value (e.g., **10**) to balance accuracy and computational cost.
   - **Neighborhood-based collaborative filtering algorithms** can handle leave-one-out cross-validation more easily.

Example:
If a ratings matrix has 100 entries and **q = 10**, each segment has 10 entries. During each of the 10 iterations:

- 90 entries are used for training.
- 10 entries are used for testing.
  The average accuracy from all 10 iterations is the final result.

5.3 Comparison with Classification Design

1. **Relation to Classification:**
   - Collaborative filtering shares similarities with classification in terms of evaluation design because both involve predicting unknown values.
   - **Classification:** Focuses on predicting a specific variable (dependent variable) based on other features.
   - **Collaborative Filtering:** Generalizes this by predicting any missing entry in a matrix, rather than just a single dependent variable.
2. **Segmentation of Data:**

- **In Classification:** Data is segmented on a **row-wise basis**, meaning some rows are used for training and others for testing.
- **In Collaborative Filtering:** Data is segmented on an **entry-wise basis**, where some individual entries in the matrix are hidden for testing, while the rest are used for training.
-

3. **Sample Selection Bias in Collaborative Filtering:**
    - Unlike classification, the hidden entries in collaborative filtering are not chosen randomly.
    - Hidden ratings usually represent items that a user has actively chosen to consume.
    - These consumed items are more likely to have **higher ratings**, as users tend to select things they expect to enjoy.
    - This creates a **sample selection bias**, meaning the hidden entries are not representative of the truly missing values (items the user hasn't consumed or considered).
4. **Impact of Bias on Evaluation:**
    - The system may perform well in predicting the hidden ratings, but this does not accurately reflect real-world performance.
    - In practice, the system will encounter a mix of highly rated items and items with unknown ratings, leading to challenges not captured during evaluation.
5. **Comparison to Classification Bias:**
    - While sample selection bias can occur in classification (e.g., biased training data), it is far more common and problematic in collaborative filtering due to the nature of the data.

Example:

Classification Scenario:

- A spam detection model predicts whether an email is spam based on labeled data.
- Rows in the dataset represent individual emails. Some rows are used for training, and others are reserved for testing.

Collaborative Filtering Scenario:

- A movie recommendation system predicts a user's rating for unwatched movies.
- The data consists of a matrix where rows represent users, columns represent movies, and entries are ratings.
- To evaluate the system, some ratings (entries) are hidden for testing. However, these hidden ratings often belong to movies the user has already chosen to watch, which are more likely to have higher ratings.

In collaborative filtering, the hidden ratings used for testing are often biased because they don't represent the broader population of movies the user hasn't interacted with, making evaluation less reflective of real-world use.

5.6. Accuracy Metrics in Offline Evaluation

Offline evaluation can be performed by measuring the accuracy of predicting rating values (e.g., with RMSE) or by measuring the accuracy of ranking the recommended items. The logic for the latter set of measures is that recommender systems often provide ranked lists of items without explicitly predicting ratings. Ranking-based measures often focus on the accuracy of only the ranks of the top-k items rather than all the items. This is particularly true in the case of implicit feedback data sets. Even in the case of explicit ratings, the ranking-based evaluations provide a more realistic perspective of the true usefulness of the recommender system because the user only views the top-k items rather than all the items.

5.6.1 Measuring the Accuracy of Ratings Prediction

Once the evaluation design for an offline experiment has been finalized, the accuracy needs to be measured over the test set. As discussed earlier, let S be the set of specified (observed) entries, and $E \subset S$ be the set of entries in the test set used for evaluation. Each entry in E is a user-item index pair of the form (u, j) corresponding to a position in the ratings matrix. The set E may correspond to the held out entries in the hold-out method, or it may correspond to one of the partitions of size |S|/q during cross-validation. Let ruj be the value of the (hidden) rating of entry $(u, j) \in E$, which is used in the test set. Furthermore, let ˆruj be the predicted rating of the entry (u, j) by the specific training algorithm being used. The entry-specific error is given by euj = ˆruj −ruj . This error can be leveraged in various ways to compute the overall error over the set E of entries on which the evaluation is performed. An example is the mean squared error, denoted by MSE:

$$MSE = \frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|}$$

Clearly, smaller values of the MSE are indicative of superior performance. The square-root of this value is referred to as the root mean squared error (RMSE), and it is often used instead of the MSE:

$$RMSE = \sqrt{\frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|}}$$

The RMSE is in units of ratings, rather than in units of squared ratings like the MSE. The RMSE was used as the standard metric for the Netflix Prize contest. One characteristic of the RMSE is that it tends to disproportionately penalize large errors because of the squared term within the summation. One measure, known as the mean-absolute-error (MAE), does not disproportionately penalize larger errors:

$$MAE = \frac{\sum_{(u,j) \in E} |e_{uj}|}{|E|}$$

Other related measures such as the normalized RMSE (NRMSE) and normalized MAE (NMAE) are defined in a similar way, except that each of them is divided by the range rmax − rmin of the ratings:

$$NRMSE = \frac{RMSE}{r_{max} - r_{min}}$$

$$NMAE = \frac{MAE}{r_{max} - r_{min}}$$

The normalized values of the RMSE and MAE always lie in the range (0, 1), and therefore they are more interpretable from an intuitive point of view. It is also possible to use these values to compare the performance of a particular algorithm over different data sets with varying scales of ratings.

RMSE versus MAE

1. **Root Mean Square Error (RMSE):**
    - **How it works:** RMSE sums up the **squared errors** and then takes the square root of the mean.
    - **Effect of Outliers:** RMSE is more sensitive to **large errors or outliers** because squaring amplifies their impact.
    - **Best for:** Applications where it's critical to minimize large errors, ensuring robustness across predictions.
2. **Mean Absolute Error (MAE):**
    - **How it works:** MAE calculates the **absolute differences** between predicted and actual values, then averages them.
    - **Effect of Outliers:** MAE treats all errors equally, regardless of size, making it less sensitive to outliers.
    - **Best for:** Applications where outliers are less critical or when a straightforward average error measure is needed.
3. **Key Considerations:**
    - **RMSE's Drawback:** It may not truly reflect the **average error**, as it emphasizes larger deviations, which can sometimes be misleading.
    - **MAE's Advantage:** Offers a more intuitive and balanced view of prediction accuracy when large errors are not disproportionately impactful.
4. **Which to Use?**
    - **RMSE:** If the application prioritizes avoiding **large prediction errors** (e.g., medical systems or financial forecasts).
    - **MAE:** When overall accuracy matters more, and **outliers are not a focus** (e.g., general recommendation systems).

Ultimately, the choice between RMSE and MAE depends on the specific needs of the application. For example:

- A movie recommendation system where a few outliers (badly predicted ratings) do not significantly affect user satisfaction may benefit from **MAE**.
- A medical diagnostic model that requires consistent accuracy across all cases may prefer **RMSE** to minimize large errors.

5.6.2 Impact of the Long Tail

1. **Impact of Popular Items:**
    - **Heavily Skewed by Popularity:** Metrics like RMSE and MAE are significantly influenced by ratings on **popular items**.
    - **Long-Tail Effect:** Ratings matrices follow a **long-tail distribution**, where most items are rated rarely while a few are rated frequently.
        - Popular items dominate the evaluation process.
        - **Long-tail items**—which are often the most profitable for merchants—are underrepresented in these metrics.

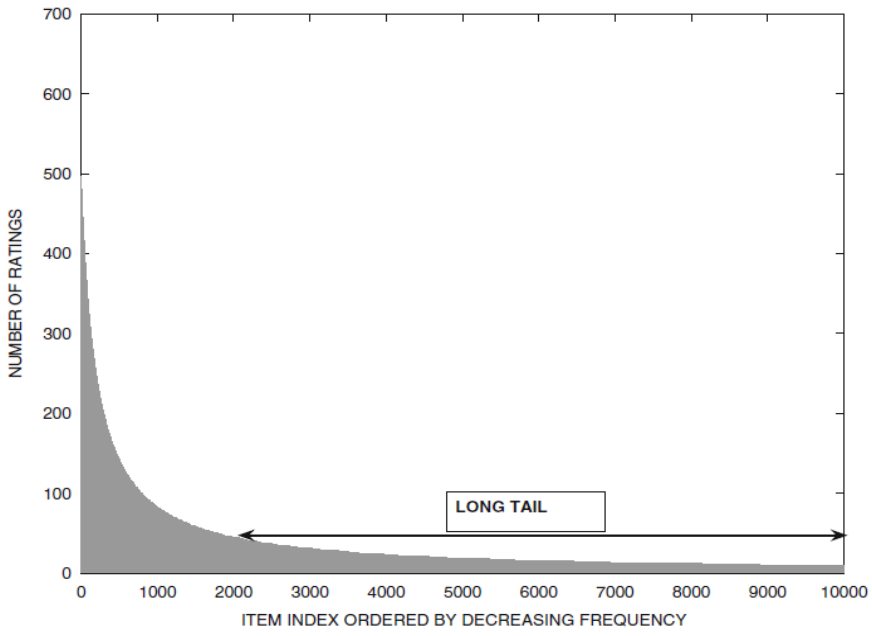


Figure: The long tail of rating frequencies

2. **Challenges with Long-Tail Items:**
    - **Sparse Data:** Long-tail items have fewer ratings, making their predictions more difficult.
    - **Unequal Weighting:** Popular items dominate accuracy calculations, leaving long-tail items with less influence on RMSE/MAE.
    - **Importance vs. Weighting:** The most important items (in terms of profit or utility) may receive the least weight in evaluations.
3. **Proposed Solutions:**

- **Item-Specific Weighting:**
  - Compute RMSE or MAE **separately for each item** and then calculate a weighted average based on:
    - **Importance:** How crucial the item is to the application.
    - **Profit or Utility:** Contribution of the item to overall business objectives.
- **User-Specific Weighting:**
  - Similar computations can be done with user-specific weights, though practical use cases for this are limited.

4. **Why This Matters:**
   - **Balanced Evaluations:** Weighting ensures that long-tail items, which are harder to predict but vital for profit, are fairly represented in the evaluation.
   - **Improved Predictions:** By addressing sparsity in long-tail items, recommendation systems can better cater to niche user preferences and business needs.

This approach helps overcome the bias towards popular items, leading to more meaningful evaluations and better alignment with real-world objectives.

5.6.2 Evaluating Ranking via Correlation

In recommendation systems, beyond predicting accurate ratings, it's crucial to assess how well the system ranks items for a user.

1. Purpose of Ranking Evaluation

- **Goal:** Ensure **highly rated items** appear higher in the ranked list compared to lower-rated items.
- **Scenario:** Consider a user u with hidden ratings for a set of items IuI_u.
  - Example: For user u, ratings for items {1, 3, 5} are hidden.
  - We evaluate how well the predicted ranking aligns with the **ground-truth ordering** of these ratings.

2. Challenges in Ranking

- **Discrete Scale of Ratings:** Ratings often use discrete values (e.g., 1–5), leading to **ties** in the ground truth.
  - Example: Two items might have the same rating (e.g., both rated 4).
  - **Solution:** Ranking measures should **not penalize** the system for ties.

3. Rank Correlation Coefficients

Two popular methods to evaluate the ranking are:

1. **Spearman's Rank Correlation Coefficient (ρ):**
   Purpose: Evaluates how well the predicted ranking aligns with the ground-truth ranking.

- **Calculation:**
  - Rank all items for both **predicted scores** and **ground-truth ratings** (from 1 to |u|).
  - Compute the **Pearson correlation coefficient** between these two ranked lists.

- The result ranges between -1 (completely opposite rankings) and +1 (perfect rankings).

2. User-Specific vs Global Spearman Coefficients

- **User-Specific Calculation:**
  Compute the Spearman coefficient for each user u, then average the results across all users.
- **Global Calculation:**
  Combine all the hidden ratings across users into one dataset and compute the Spearman coefficient for the entire set.

3. Handling Ties with Tie-Corrected Spearman

- **Challenge:**
  Ground-truth rankings often have ties (e.g., multiple items rated the same).

  - Randomly breaking ties introduces noise, reducing the reliability of the evaluation.
- **Solution:**
  Use **average ranks** for tied items instead of arbitrary tie-breaking.

  Example:
  Suppose the ground-truth ratings are as follows for 4 items:

  - Item 1: Rating = 5
  - Item 2: Rating = 5
  - Item 3: Rating = 4
  - Item 4: Rating = 3
- Instead of assigning ranks as {1, 2, 3, 4}, use:

  - Item 1: Rank = 1.5
  - Item 2: Rank = 1.5
  - Item 3: Rank = 3
  - Item 4: Rank = 4

4. Why Use Tie-Corrected Spearman?

- It ensures a fair and consistent ranking evaluation.
- It reduces noise introduced by arbitrary tie-breaking, leading to more accurate correlation measurement.

By addressing ties effectively, **tie-corrected Spearman rank correlation** provides a robust and interpretable metric for ranking evaluation in recommender systems.

2. **Kendall's Tau (τ):**

   For each pair of items j, k ∈ Ii, the following credit C(j, k) is computed by comparing the predicted ranking with the ground-truth ranking of these items:

$$C(j,k) = \begin{cases} +1 & \text{if items } j \text{ and } k \text{ are in the same relative order in ground-truth ranking and predicted ranking (concordant)} \\ -1 & \text{if items } j \text{ and } k \text{ are in a different relative order in ground-truth ranking and predicted ranking (discordant)} \\ 0 & \text{if items } j \text{ and } k \text{ are tied in either the ground-truth ranking or predicted ranking} \end{cases}$$

Then, the Kendall rank correlation coefficient τ, which is specific to user u, is computed as the average value of C(j, k) over all the |Iu|(|Iu| − 1)/2 pairs of test items for user i:

$$\tau_u = \frac{\sum_{j<k} C(j,k)}{|I_u| \cdot (|I_u| - 1)/2}$$

A different way of understanding the Kendall rank correlation coefficient is as follows:

$$\tau_u = \frac{\text{Number of concordant pairs} - \text{Number of discordant pairs}}{\text{Number of pairs in } I_u}$$

- **Purpose:** Compares the number of **concordant pairs** (items ranked correctly relative to each other) to **discordant pairs** (items ranked incorrectly).
- **Calculation:**
  - Concordant pair: If item A is ranked above B in both ground truth and predicted rankings.
  - Discordant pair: If A and B are ranked oppositely.
- **Output:** Similar to Spearman's ρ, values range from -1 to +1.

The **Normalized Distance-Based Performance Measure (NDPM)** is another method used to evaluate ranking performance, particularly in recommender systems. Here's an overview of NDPM:

- NDPM measures the distance between the predicted ranking and the ground-truth ranking of items.
- It provides a normalized score, where smaller values indicate better alignment between predicted and true rankings.

Calculation Steps

1. **Identify Ground-Truth Ranking:** Rank items based on their true ratings (or relevance scores).
2. **Compute Predicted Ranking:** Rank items as predicted by the recommender system.
3. **Measure Distance:** Calculate the distance between each item's position in the ground-truth ranking and the predicted ranking.

4. **Normalize:** Scale the distances to produce a score that accounts for the number of items and ranking depth.

Interpretation of NDPM

- **Low NDPM Value:**
  Indicates a strong alignment between predicted and actual rankings (better recommendation quality).
- **High NDPM Value:**
  Suggests poor alignment, with the recommender failing to rank items close to their true importance.

Why Rank Correlation is Important

- **Practical Utility:** Most recommendation systems focus on presenting a **top-kk** list to users.
- **Alignment with Preferences:** Rank correlation helps evaluate how well the system prioritizes items users are likely to value.

By leveraging these measures, ranking evaluations ensure that recommendations are not only accurate but also **meaningful in their order** of presentation.

5.6.3 Evaluating Ranking via Utility

Utility-based evaluation methods measure how useful a recommender system's ranking is to the user by combining the system's predicted ranking with the ground-truth rating of items. These methods aim to reflect the actual value a user might derive from a recommendation list, particularly emphasizing the importance of highly-ranked items.

Key Principles

1. **Short Recommendation Lists**:
   Most utility in a recommendation system comes from the top-ranked items, as users typically focus only on the first few recommendations.

2. **Ground-Truth Rating**:
   Utility depends not only on the rank of an item in the recommendation list but also on its ground-truth rating or relevance.

3. **Weakness of RMSE**:
   RMSE does not prioritize top-ranked items, treating errors across all ranks equally. Even small improvements in RMSE (e.g., 1%) can cause significant changes (e.g., 15%) in the top-rated items, which are most relevant to users.

Components of Utility-Based Evaluation

1. **Rating-Based Utility**:
   - For any item j rated by a user u, its rating-based utility is calculated as:

$$\text{Utility}_{\text{rating}} = \max\{r_{uj} - C_u, 0\}$$

- ruj: Ground-truth rating of item j for user u.
- Cu: Break-even (neutral) rating value for user u, often set to the user's average rating.
- This ensures that only items rated above the neutral value contribute positively to the utility.

2. **Ranking-Based Utility**:

   - Items ranked higher in the recommendation list have greater utility because they are more likely to be noticed and selected by the user.
   - The ranking-based utility of an item j is:

$$\text{Utility}_{\text{rank}} = 2^{-\frac{(v_j - 1)}{\alpha}}$$

- vj: The rank of item j in the recommended list.
- α : A half-life parameter that controls the rate of exponential decay in utility.
- Moving down the ranking by α positions reduces utility by half.

Final Utility Calculation

The total utility of an item j is influenced by both its rating-based utility and ranking-based utility. Items with higher ground-truth ratings and higher ranks contribute more to the overall utility. This emphasizes the importance of the top-ranked items in the evaluation process.

Advantages of Utility-Based Evaluation

1. **Focus on Top Items**:
   These methods prioritize top-ranked items, which are the most relevant to users and more likely to be consumed.

2. **Realistic User Behavior Modeling**:
   Utility decreases with rank, reflecting that users rarely scroll through lower-ranked items.

3. **Better Alignment with User Goals**:
   Unlike RMSE, utility-based methods align more closely with the practical objectives of a recommender system by focusing on the items that users are most likely to interact with.

Utility-based methods are particularly well-suited for systems where ranking accuracy at the top of the list is critical, such as e-commerce platforms, streaming services, and content discovery tools. By considering both the ground-truth ratings and ranking positions, these methods provide a holistic evaluation of recommendation quality.

The **utility F(u, j)** of item $j \in I_u$ to user u is defined as the product of the rating-based and ranking-based utility values:

$$F(u,j) = \frac{\max\{r_{uj} - C_u, 0\}}{2^{(v_j-1)/\alpha}}$$

The **R-score**, which is specific to user u, is the sum of $F(u, j)$ over all the hidden ratings in I_u:

$$\text{R-score}(u) = \sum_{j \in I_u} F(u,j)$$

Note that the value of vj can take on any value from 1 to n, where n is the total number of items. However, in practice, one often restricts the size of the recommended list to a maximum value of L. One can therefore compute the R-score over a recommended list of specific size L instead of using all the items, as follows:

$$\text{R-score}(u) = \sum_{j \in I_u, v_j \leq L} F(u,j)$$

The idea here is that ranks below L have no utility to the user because the recommended list is of size L. This variation is based on the principle that recommended lists are often very short compared to the total number of items. The **overall R-score** may be computed by summing this value over all the users.

$$\text{R-score} = \sum_{u=1}^{m} \text{R-score}(u)$$

The exponential decay in the utility implies that users are only interested in top-ranked items, and they do not pay much attention to lower-ranked items. This may not be true in all applications, especially in news recommender systems, where users typically browse multiple items lower down the list of recommended items. In such cases, the discount rate should be set in a milder way.

An example of such a measure is the **discounted cumulative gain (DCG).** In this case, the discount factor of item j is set to $\log_2(v_j+1)$, where v_j is the rank of item j in the test set I_u. Then, the discounted cumulative gain is defined as follows:

$$DCG = \frac{1}{m} \sum_{u=1}^{m} \sum_{j \in I_u} \frac{g_{uj}}{\log_2(v_j + 1)}$$

Here, guj represents the utility (or gain) of the user u in consuming item j. Typically, the value of guj is set to an exponential function of the relevance (e.g., non-negative ratings or user hit rates):

$$g_{uj} = 2^{rel_{uj}} - 1$$

Here, rel_{uj} is the ground-truth relevance of item j for user u, which is computed as a heuristic function of the ratings or hits. In many settings, the raw ratings are used. It is common to compute the

discounted cumulative gain over a recommendation list of specific size L, rather than using all the items:

$$DCG = \frac{1}{m}\sum_{u=1}^{m}\sum_{j \in I_u, v_j \leq L}\frac{g_{uj}}{\log_2(v_j+1)}$$

The basic idea is that recommended lists have size no larger than L. Then, the normalized discounted cumulative gain (NDCG) is defined as ratio of the discounted cumulative gain to its ideal value, which is also referred to as ideal discounted cumulative gain (IDCG).

$$NDCG = \frac{DCG}{IDCG}$$

The ideal discounted cumulative gain is computed by repeating the computation for DCG, except that the ground-truth rankings are used in the computation.

Average reciprocal hit rate (ARHR) - This measure is designed for implicit feedback data sets, in which each value of $ruj \in \{0, 1\}$. Therefore, a value of $ruj = 1$ represents a "hit" where a customer has bought or clicked on an item. A value of $ruj = 0$ corresponds to a situation where a customer has not bought or clicked on an item. In this implicit feedback setting, missing values in the ratings matrix are assumed to be 0. In this case, the rank-based discount rate is $1/vj$, where vj is the rank of item j in the recommended list, and the item utility is simply the hidden "rating" value $ruj \in \{0, 1\}$.

Note that the discount rate is not as rapid as the R-score metric, but it is faster than DCG. Therefore, the combined utility of an item is given by r_{uj}/v_j . This expression represents the contribution of item $j \in Iu$ to the utility. Then, the ARHR metric for the user i is defined by summing up these values over all the hidden items in Iu:

$$ARHR(u) = \sum_{j \in I_u}\frac{r_{uj}}{v_j}$$

It is also possible to define the average reciprocal hit-rate for a recommended list of size L by adding only those utility values for which $vj \leq L$.

$$ARHR(u) = \sum_{j \in I_u, v_j \leq L}\frac{r_{uj}}{v_j}$$

The **hit-rate (HR)** is simply the fraction of users for which the correct answer is included in the recommendation list of length *L*. The disadvantage of the hit-rate is that it gives equal importance to a hit, irrespective of its position in the recommended list.

The ***mean average precision (MAP)***, which computes the fraction of relevant items in a recommended list of length *L* for a given user. Various equally spaced values of *L* are used, and the precision is averaged over these recommendation lists of varying lengths. The resulting precision is then averaged over all the users.

5.4 Evaluating Ranking via Receiver Operating Characteristic (ROC)

Ranking methods are often used to evaluate recommender systems, especially when considering implicit feedback datasets like sales, clicks, or movie views, where actions such as purchases or views can be treated as binary outcomes (e.g., consumed or not consumed). These methods are effective for evaluating systems where the ground-truth is binary (i.e., whether an item was consumed or not).

1. **Binary Ground-Truth**:
    - In implicit feedback data, the ground-truth for items is binary: items that are consumed are marked as relevant (true positives), while unconsumed items are treated as non-relevant.
2. **Evaluation of Ranked Lists**:
    - Recommender systems generate ranked lists of items. The goal is to evaluate how many of the recommended items are actually relevant (i.e., consumed).
3. **Trade-off Between Recall and Precision**:
    - The performance of a ranking algorithm is influenced by the number of items recommended. A larger list may include more relevant items, but it also risks recommending irrelevant items (false positives). A smaller list may miss relevant items (false negatives).
    - The trade-off between precision (correct recommendations) and recall (capturing all relevant items) can be measured using two key metrics:

Precision-Recall Curve and **Receiver Operating Characteristic (ROC) Curve**.

Precision-Recall vs. ROC Curve

- **Precision**: The percentage of recommended items that are relevant (ground-truth positive).
- **Recall**: The percentage of relevant items that are captured by the recommender system.

ROC curves and precision-recall curves are used to evaluate how well the system is performing in capturing relevant items, as they show the trade-off between **true positives** (relevant items recommended) and **false positives** (irrelevant items recommended), depending on the size of the recommendation list.

How ROC Works

1. **Ranking and Scoring**:
    - The recommender system ranks all items using a numerical score, which can be thought of as the algorithm's prediction of relevance. The top items in the ranked list are recommended.
2. **Varying List Size**:
    - By adjusting the number of items recommended (i.e., varying the size of the ranked list), the system's performance can be examined in terms of:
        - **True Positives**: Relevant items correctly recommended.
        - **False Positives**: Irrelevant items incorrectly recommended.
        - **False Negatives**: Relevant items that are missed.
3. **Evaluation Metrics**:
    - If the list is too small, many relevant items will be missed (false negatives).

- If the list is too large, there will be many irrelevant items included (false positives).

Practical Application

- ROC curves are commonly used in rare class detection, outlier analysis, and information retrieval. They provide a visual representation of the trade-off between correctly identified relevant items and the irrelevant ones recommended, helping to fine-tune the system for better performance.
- By adjusting the size of the recommendation list, one can visualize how changes affect the system's ability to recommend relevant items and avoid spurious ones.

In summary, ROC and precision-recall curves are powerful tools for evaluating ranking-based recommender systems. They help assess the balance between recommending too many irrelevant items and missing relevant ones, with the goal of optimizing the size and quality of the recommended list.

Assume that one selects the top-t set of ranked items to recommend to the user. For any given value t of the size of the recommended list, the set of recommended items is denoted by $S(t)$. Note that $|S(t)| = t$. Therefore, as t changes, the size of $S(t)$ changes as well. Let G represent the true set of relevant items (ground-truth positives) that are consumed by the user. Then, for any given size t of the recommended list, the *precision* is defined as the percentage of recommended items that truly turn out to be relevant (i.e., consumed by the user).

$$Precision(t) = 100 \cdot \frac{|\mathcal{S}(t) \cap \mathcal{G}|}{|\mathcal{S}(t)|}$$

The *recall* is correspondingly defined as the percentage of *ground-truth* positives that have been recommended as positive for a list of size t.

$$Recall(t) = 100 \cdot \frac{|\mathcal{S}(t) \cap \mathcal{G}|}{|\mathcal{G}|}$$

While a natural trade-off exists between precision and recall, this trade-off is not necessarily monotonic. In other words, an increase in recall does not always lead to a reduction

in precision. One way of creating a single measure that summarizes both precision and recall is the $F1$-measure, which is the harmonic mean between the precision and the recall.

$$F_1(t) = \frac{2 \cdot Precision(t) \cdot Recall(t)}{Precision(t) + Recall(t)}$$

While the $F1(t)$ measure provides a better quantification than either precision or recall, it is still dependent on the size t of the recommended list and is therefore still not a complete representation of the trade-off between precision and recall.

A second way of generating the trade-off in a more intuitive way is through the use of the ROC curve. The *true-positive rate*, which is the same as the recall, is defined as the percentage of ground-truth positives that have been included in the recommendation list of size t.

$$TPR(t) = Recall(t) = 100 \cdot \frac{|\mathcal{S}(t) \cap \mathcal{G}|}{|\mathcal{G}|}$$

The false-positive rate $FPR(t)$ is the percentage of the falsely reported positives in the recommended list out of the ground-truth negatives (i.e., irrelevant items not consumed by the user). Therefore, if U represents the universe of all items, the ground-truth negative set is given by $(U - G)$, and the falsely reported part in the recommendation list is $(S(t) - G)$. Therefore, the false-positive rate is defined as follows:

$$FPR(t) = 100 \cdot \frac{|\mathcal{S}(t) - \mathcal{G}|}{|\mathcal{U} - \mathcal{G}|}$$

The false-positive rate can be viewed as a kind of "bad" recall, in which the fraction of the ground-truth negatives (i.e., items not consumed), which are incorrectly captured in the recommended list $S(t)$, is reported. The ROC curve is defined by plotting the $FPR(t)$ on the X-axis and $TPR(t)$ on the Y -axis for varying values of t. In other words, the ROC curve plots the "good" recall against the "bad" recall. Note that both forms of recall will be at 100% when $S(t)$ is set to the entire universe of items.

Therefore, the end points of the ROC curve are always at (0, 0) and (100, 100), and a random method is expected to exhibit performance along the diagonal line connecting these points. The *lift* obtained above this diagonal line provides an idea of the accuracy of the approach. The area under the ROC curve provides a concrete quantitative evaluation of the effectiveness of a particular method.

Insights from Ranking Evaluation Measures

Example Scenario

Consider a scenario with **100 items**, where only **5 items are truly relevant** to the user. Two algorithms, **A** and **B**, rank all 100 items. The goal is to evaluate how well these algorithms perform by examining their ability to rank the relevant items higher in the list.

Table 7.1: Rank of ground-truth positive instances

Algorithm	Rank of items that are truly used (ground-truth positives)
Algorithm A	1, 5, 8, 15, 20
Algorithm B	3, 7, 11, 13, 15
Random Algorithm	17, 36, 45, 59, 66
Perfect Oracle	1, 2, 3, 4, 5

ROC and Precision-Recall Curves

- **ROC Curves**: These curves are generated by plotting the **true positive rate** (TPR) against the **false positive rate** (FPR) for different thresholds, where a lower rank (higher relevance) corresponds to a positive recommendation. The ROC curve increases monotonically, meaning that as more items are included in the recommendation list, the true positive rate increases while the false positive rate may also increase.

- **Precision-Recall Curves**: These are not monotonically increasing, but they show how the **precision** (proportion of relevant items in the recommendation list) and **recall** (the proportion of relevant items captured by the list) change as the list size increases.

The key takeaway from these curves is:

- **ROC curves** are often preferred due to their clearer interpretation and monotonicity.

- **Precision-Recall curves** may offer more detailed insights, but are less straightforward to interpret, especially in systems with imbalanced data.

Interpreting the Curves

- **Oracle Algorithm**: This algorithm ranks the correct top 5 items highest. As expected, it performs best on both ROC and precision-recall curves.
- **Random Algorithm**: This algorithm ranks the items randomly, leading to a worst-case performance on both curves.
- **Algorithms A and B**: These algorithms show different performance patterns:
    - **Algorithm A** ranks three relevant items highly but ranks the other two poorly, resulting in better performance early in the ROC curve.
    - **Algorithm B** ranks all relevant items relatively well, but none are ranked as highly as Algorithm A's top-ranked items. This means Algorithm B performs better at the later part of the ROC curve but worse at the early part.

Area Under the ROC Curve (AUC)

The **AUC** of the ROC curve can provide an overall measure of the algorithm's effectiveness. However, not all parts of the ROC curve are equally important. This is because users typically only interact with the top-ranked items, so the earlier parts of the curve (which represent the top recommendations) are more critical for real-world performance.

Customer-Specific vs. Global ROC Curves

- **Customer-Specific ROC Curves**: These curves are created by evaluating each user's recommended list separately.
- **Global ROC Curves**: These are generated by ranking all possible user-item pairs across all users, considering the predicted ratings or affinities.

Which Ranking Measure Is Best?

- **ROC Curves**: While useful for evaluating ranking performance, ROC curves treat all ranks equally and may not reflect the actual user experience. The relative importance

of higher-ranked items (those at the top of the recommendation list) is often much greater than lower-ranked items.

- **Utility-Based Measures**: Measures like **NDCG** (Normalized Discounted Cumulative Gain) better capture the real-world utility of rankings. These measures give more weight to the **higher-ranked items**, aligning better with the user's focus on top recommendations. Unlike ROC, which treats all items equally, utility-based measures recognize that items at the top of the list are far more important than those lower down.

While ROC curves are widely used for evaluating ranking algorithms, they may not fully capture the user experience, particularly in systems where the **top-ranked items** are far more important than those lower down the list. In these cases, **utility-based measures** like **NDCG** provide a more accurate reflection of the system's true performance by focusing on the relevance of higher-ranked items.

5.7 Limitations of Evaluation Measures in Recommender Systems

1. Selection Bias and Missing Data

In recommender systems, missing ratings are **not random**. Users tend to rate more popular items or items they are more familiar with. As a result, certain patterns emerge, leading to a **selection bias**. The ratings on popular items differ from those on less popular items, especially those in the **long tail** of the item distribution.

For example, more popular items are typically rated higher due to their content or visibility, which skews the accuracy of recommendation algorithms. This is an inherent problem in evaluating recommender systems using accuracy-based measures because:

- **Ratings on popular items** might be more biased (due to higher visibility).
- **Items in the long tail** might be underrepresented, despite their importance.

This bias is known as **Missing Not At Random (MNAR)** or **selection bias**, where the missing ratings are **not randomly distributed** but influenced by the likelihood of a user rating certain items.

2. Impact on Evaluation

- **Ground truth bias**: Simulated missing ratings (via cross-validation or hold-out) may not reflect the true nature of the ratings that would occur if users were forced to rate all items. This discrepancy can lead to **incorrect evaluations** of recommender system algorithms.

- **Popularity bias**: Popularity-based models might perform well in terms of revenue generation (e.g., recommending highly rated popular items) but not on the basis of **randomly missing ratings**. This creates a mismatch between **evaluation measures** and **real-world performance**, where the **long tail** items (less popular but valuable) are often underrepresented.

3. Solutions to Bias in Evaluation

- **Non-random selection of ratings**: Instead of selecting ratings randomly, models can be designed to select test ratings based on their likelihood of being rated in the future.

- **Temporal division of data**: Instead of randomly splitting data for training and testing, the system can use **recent ratings** as part of the test data. This reflects a more **real-world setting**, where user preferences evolve over time.

- **Bias modeling**: Some recent methods model the bias in missing ratings distributions to adjust the evaluation metrics accordingly. However, this introduces complexities and could favor algorithms that already use similar models for rating predictions, potentially leading to **evaluation gaming**.

4.Temporal Evolution of User Interests

User interests evolve over time. The preferences reflected in **historical ratings** might not be predictive of future behavior. For example, a **hold-out set** (a set of ratings held back for testing) may not capture shifts in user preferences, leading to an **evaluation mismatch** with future performance. This is where **temporal validation**—splitting the data based on time—becomes crucial, as it simulates the changing user behavior over time.

5.Gaming the Evaluation Process

The **Netflix Prize** contest highlighted the issue of **evaluation gaming**. When user-item pairs in the test set are **specified in advance**, algorithms can incorporate this knowledge (like implicit feedback) to gain an unfair advantage. This is unrealistic in real-world scenarios, where algorithms don't have access to future test data. This **gaming** can lead to the **overfitting** of algorithms that perform well only because they exploit this additional knowledge.

Possible solutions include:

- **Not specifying the test set coordinates**: This would prevent algorithms from using knowledge about the future ratings in the test set.
- **Incorporating spurious unrated entries**: Adding fake ratings in the test set prevents algorithms from using prior knowledge of the test entries and forces them to make more general predictions.

Key Takeaways

1. **Selection bias** due to users rating more popular items can distort accuracy-based evaluations.
2. The assumption that **ratings are missing randomly** does not hold in real-world systems, leading to **evaluation errors**.
3. The use of **temporal methods** for evaluation, where training and test data are divided based on time, helps simulate real-world conditions where user preferences evolve.
4. The risk of **evaluation gaming** arises when algorithms leverage information about test data to improve performance, making evaluations unrealistic.

ISBN: 978-81-989720-3-3

SUMMARY – UNIT 5

Evaluating Recommender Systems

Evaluating the effectiveness of recommender systems is essential for understanding how well different algorithms perform in recommending relevant items. There are various ways to measure the performance of these systems, with each method addressing different aspects of the recommender process. Below are some common evaluation approaches:

1. Conversion Rate

- **Definition**: This metric measures the percentage of recommended items that are actually used or consumed by users.
- **Methods**: Conversion rates can be evaluated through **user studies** or **online studies**, where a group of users interacts with the system and their behavior is tracked. However, these studies are challenging due to the need for large user groups and relevant infrastructure.
- **Limitation**: Relying solely on conversion rates may not always reflect long-term performance, as maximizing short-term accuracy doesn't guarantee sustained success.

2. Offline Evaluation Methods

- **Advantages**: Offline methods allow evaluation using **historical data** without needing real-time user interaction. This can be advantageous for comparing multiple algorithms across different data sets.
- **Risk**: Simply maximizing **accuracy** in offline evaluations may not lead to long-term success in practical applications. For instance, a system might appear accurate in the short term but fail to keep users engaged over time.

3. Evaluation Criteria

To thoroughly assess a recommender system, a variety of criteria are considered:

- **Coverage**: This refers to the percentage of items from the catalog that the recommender system can effectively recommend. It ensures that the system isn't just recommending a small subset of items, but it can potentially cover the entire catalog.
- **Novelty**: This measure evaluates how well the system recommends items that the user may not have encountered otherwise. It assesses how "new" or unexpected the recommendations are for the user.
- **Serendipity**: A related metric to novelty, **serendipity** measures the **pleasant surprises** in the recommendations—items that the user may not have thought to explore but end up finding interesting.

- **Stability**: This metric evaluates how consistent the recommendations are over time. If a user's preferences change slightly, the recommender should still suggest relevant items, reflecting **stability**.

- **Scalability**: This refers to the ability of the recommender system to handle an increasing amount of data, including users, items, and interactions, without compromising performance.

4. Out-of-Sample Evaluation

To avoid **bias** in the evaluation process, an **out-of-sample evaluation** approach is critical. In collaborative filtering applications, this can be achieved through techniques like:

- **Hold-out**: A subset of the data is reserved for testing, while the rest is used for training the recommender system.

- **Cross-validation**: The data is split into multiple subsets, and the system is tested on each one while being trained on the others.

This ensures that the system is evaluated on data it has not seen before, providing a more realistic measure of its performance.

5. Error Metrics

To quantify the accuracy of recommender systems, various error metrics are used:

- **MAE (Mean Absolute Error)**: Measures the average absolute difference between predicted and actual ratings.

- **MSE (Mean Squared Error)**: Similar to MAE but penalizes larger errors more heavily due to squaring the differences.

- **RMSE (Root Mean Squared Error)**: The square root of MSE, often used as it is in the same units as the ratings, making it easier to interpret.

6. Usage-Based Measures

In addition to accuracy measures, recommender systems can be evaluated based on **usage-based metrics**. These focus on the system's ability to recommend items that are actually relevant to the user. Common metrics include:

- **Precision**: Measures the proportion of recommended items that are relevant (true positives).

- **Recall**: Measures the proportion of relevant items that are actually recommended.

- **F1-Score**: The harmonic mean of precision and recall, which provides a single score that balances both metrics. This is useful when precision and recall need to be traded off.

ISBN: 978-81-989720-3-3

7. Ranking Measures

For **ranking algorithms**, where the goal is to rank items from most relevant to least relevant, alternative evaluation methods are used:

- **Rank Correlation**: Measures the degree to which the ranks predicted by the system match the actual ranks of items, typically using metrics like **Spearman's Rank Correlation** or **Kendall's Tau**.
- **Utility-based Measures**: These consider the **overall utility** of recommendations, which may weigh the ranking of items differently based on factors like user preferences or item importance.

About Author

N. Nancy Chitra Thilaga, M.E. in Applied Electronics, is an Assistant Professor at Grace College of Engineering, Thoothukudi. She brings a unique blend of academic insight and industry experience, having spent four years in the IT sector before transitioning to academia. Her exposure to real-world technological challenges enhances both her teaching and writing.

With a strong academic foundation and a deep interest in emerging technologies, she delivers content that is both conceptually clear and practically relevant. Her focus on curriculum-aligned material and application-oriented perspectives makes this book a valuable contribution to the academic literature in the fields of recommender systems and artificial intelligence.

Nancy Chitra Thilaga N

Assistant Professor

nancychitrathilaga@grace.edu.in

Dr.BGR Publications

ISBN 978-81-989720-3-3

9 788198 972033