**Module 1**

Data sparsity and the cold-start problem

In real-world applications, of course, the rating matrices tend to be very sparse, as customers typically provide ratings for (or have bought) only a small fraction of the catalog items. *Approaches to deal with the cold-start and data sparsity problems*

1. One straightforward option for dealing with this problem is to ***exploit additional information about the users***, such as gender, age, education, interests, or other available information that can help to classify the user. The set of similar users (neighbors) is thus based not only on the analysis of the explicit and implicit ratings, but also on information external to the ratings matrix.
2. Graph-based method:

Spreading activation in Bipartite graph of users and items

A bipartite graph is a type of graph with two types of nodes:

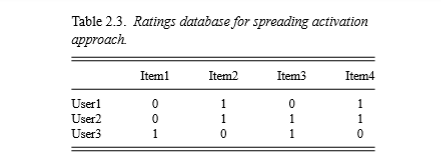
1. **Users**: People interacting with the system.
2. **Items**: Products, movies, books, or other entities to recommend.

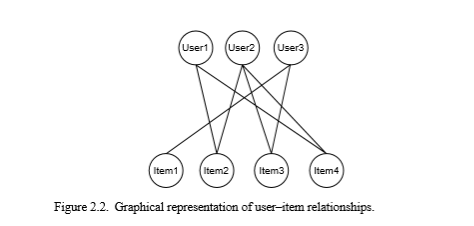
Edges (connections) in this graph represent interactions between users and items, like a user watching a movie or buying a product.

Spreading activation is a method to propagate "influence" or "scores" through the graph to infer relationships indirectly. Here's how it works:

1. Start with a **source node** (e.g., a new user or item).
2. Assign an initial score or activation to that node.
3. Spread this activation to connected nodes (neighbors) based on predefined rules.
4. Repeat the process to further propagate the scores through the graph.

The main idea of this approach is to exploit the supposed “transitivity” of customer tastes and thereby augment the matrix with additional information. Consider the user-item relationship graph in the following Figure which can be inferred from the binary ratings matrix Table.





A 0 represents a missing rating. In graph analysis problem, recommendations are determined by determining paths between users and items. In a standard user-based or item-based CF approach, paths of length 3 will be considered – that is, Item3 is relevant for User1 because there exists a three-step path (User1–Item2–User2–Item3) between them. Because the number of such paths of length 3 is small in sparse rating databases, the idea is to also consider longer paths (indirect associations) to compute recommendations. Here, computation of these distant relationships is expensive.

A comparison with the standard user-based and item-based algorithms shows that the quality of the recommendations can be signiﬁcantly improved with the proposed technique based on indirect relationships, in particular when the ratings matrix is sparse. Also, for new users, the algorithm leads to measurable performance increases when compared with standard collaborative ﬁltering techniques.

1. Default voting

Default voting is a method used in recommendation systems to handle **sparse rating databases**, where many items are unrated. Sparse data creates challenges in accurately computing similarity between users or items.

Default voting introduces **artificial default ratings** for items that:

1. **Only one user has rated.**
2. **Are unrated by both users**, in some cases.

By assigning default values to these unrated items, the similarity measure becomes more stable and less influenced by sparse data.

* Assign a **default rating** (e.g., average rating, neutral score, or a specific value like 3 on a 1–5 scale) to unrated items.
* Use these default ratings alongside actual ratings to compute similarity.

**Challenges:**

* Choosing the right **default value** is critical:
  + Too high or too low values can distort similarity.
  + Neutral values (e.g., average ratings) often work best.
* Can increase computational cost.

1. Use both user similarities or item similarities

Combine both user-based and item-based similarities to improve prediction accuracy, rather than relying on just one similarity type.

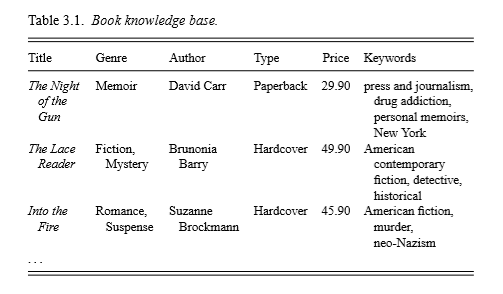
**Content-based recommendation**

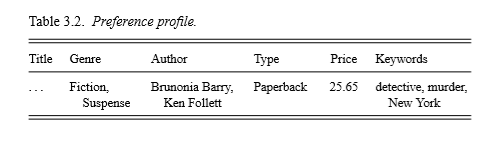
Content-based recommendation systems are a popular and widely used approach to provide personalized recommendations to users. These systems are based on the idea that a user's preferences can be predicted based on their previous interactions with items, such as their viewing and purchasing history.

Content representation and content similarity

The simplest way to describe catalog items is to maintain an explicit list of features for each item (also often called attributes, characteristics, or item proﬁles). For a book recommender, one could, for instance, use the genre, the author’s name, the publisher, or anything else that describes the item and store this information in a relational database system. When the user’s preferences are described in terms of his or her interests using exactly this set of features, the recommendation task consists of matching item characteristics and user preferences.

Content-based recommendation systems aim to provide **personalized recommendations** by analyzing the features of items that users have interacted with in the past. They assume that a user will prefer items similar to those they already like.





Consider the above example (table 3.1) in which books are described by characteristics such as title, genre, author, type, price, or keywords. Let us further assume that Alice’s preferences are captured in exactly the same dimensions as in table 3.2.

The similarity or overlap of the involved keywords. As a typical similarity metric that is suitable for multivalued characteristics, we could, for example, rely on the *Dice coefﬁcient* as follows: If every book Bi is described by a set of keywords keywords(Bi), the Dice coefﬁcient measures the similarity between books bi and bj as

D:\simi\MESCE\23-24 even\s6 ADS\IMAGES\dice.PNG

### The Vector Space Model and TF-IDF in Recommendation Systems

The vector space model and TF-IDF

The content of a document can be encoded in such a keyword list in different ways. In a ﬁrst, and very naive, approach(Boolean approach), one could set up a list of all words that appear in all documents and describe each document by a Boolean vector, where a 1 indicates that a word appears in a document and a 0 that the word does not appear. If the user proﬁle is described by a similar list (1 denoting interest in a keyword), document matching can be done by measuring the overlap of interest and document content.

The problems with such a simple approach are:

* The simple encoding is based on the assumption that every word has the same importance within a document, although it seems intuitive that a word that appears more often is better suited for characterizing the document.
* A larger overlap of the user proﬁle and a document will naturally be found when the documents are longer. As a result, the recommender will tend to propose long documents.

To solve this, documents are typically described using the TF-IDF encoding format is used.

TF-IDF is an established technique from the ﬁeld of information retrieval and stands for term frequency-inverse document frequency. Text documents can be TF-IDF encoded as vectors in a multidimensional Euclidian space. The space dimensions correspond to the keywords (also called terms or tokens) appearing in the documents. The coordinates of a given document in each dimension (i.e., for each term) are calculated as a product of two submeasures: term frequency and inverse document frequency.

**Term frequency** describes *how often a certain term appears in a document* (assuming that important words appear more often.

We search for the normalized term frequency value TF(i,j) of keyword i in document j. Letfreq(i,j) be the absolute number of occurrences of i in j. maxOthers(i,j) denotes the number of occurrences of other terms. Finally, calculate TF(i,j) as in

D:\simi\MESCE\23-24 even\s6 ADS\IMAGES\tf.PNG

**Inverse document frequency** is the second measure that is combined with term frequency. It aims at *reducing the weight of keywords that appear very often in all documents* therefore be given to words that appear in only a few documents. Let N be the number of all recommendable documents and n(i) be the number of documents from N in which keyword i appears. The inverse document frequency for i is typically calculated as

D:\simi\MESCE\23-24 even\s6 ADS\IMAGES\log.PNG

The combined TF-IDF weight for a keyword i in document j is computed as the product of these two measures:

D:\simi\MESCE\23-24 even\s6 ADS\IMAGES\tf-idf.PNG

**Improving the vector space model/limitations**

TF-IDF vectors are typically large and very sparse. To make them more compact and to remove irrelevant information from the vector, additional techniques can be applied.

* Stop words and stemming.

A straightforward method is to remove so-called stop words. In the English language these are, for instance, prepositions and articles such as “a”, “the”, or “on”, which can be removed from the document vectors because they will appear in nearly all documents. Another commonly used technique is called stemming or conﬂation, which aims to replace variants of the same word by their common stem (root word). The word “stemming” would, for instance, be replaced by “stem”, “went” by “go”, and so forth.

Stemming procedures are commonly implemented as a combination of morphological analysis using, for instance, Porter’s sufﬁx-stripping. . But here there is a problem. For example, both the terms university and universal are stemmed to univers, which may lead to an unintended match of a document with the user proﬁle.

* Size cutoffs. Another straightforward method to reduce the size of the document representation and hopefully remove “noise” from the data is to use only the n most informative words. Feature selection” can also be applied for determining the most informative keywords.
* Phrases. A further possible improvement with respect to representation accuracy is to use “phrases as terms”, which are more descriptive for a text than single words alone. Phrases, or composed words such as “United Nations”, can be encoded as additional dimensions in the vector space.

Limitations. The described approach of extracting and weighting individual keywords from the text has another important limitation: it does not take into account the context of the keyword and, in some cases, may not capture the “meaning” of the description correctly. Consider the following simple example. A free-text description of a steakhouse used in a corresponding recommender system might state that “there is nothing on the menu that a vegetarian would like”. In this case, in an automatically generated feature vector, the word vegetarian will most probably receive a higher weight than desired and produce an unintended match with a user interested in vegetarian restaurants.