# Cold Start Problems in Recommender Systems

**Felicia Grace Ravichandran  Nikhil Gupta  Shreya Shivpuje  Tsu-Hao Fu**
{fravicha,gupta883,sshivpu,fu390}@purdue.edu

## Abstract

A recommender system aims to provide personalized recommendations to users on content, such as what product to buy, which music to listen to, or which online news article to read. The cold start problem in recommendation systems encompasses challenges when dealing with new users or items. To find solutions, data must be gathered, either explicitly from users or implicitly from existing data. Addressing the cold start problem requires accurate recommendations, bias reduction, adaptability, and diversity to ensure meaningful suggestions for new users and items in the recommendation system. We aim to study the recommender system and assess the performance of standard techniques in tackling the cold start problem. Our experiments offer insights into the relative performance of these techniques.

## 1 Introduction and Motivation

Recommender systems tailor suggestions to users when picking products, music, or online articles to read. They infer user preferences explicitly or implicitly to present items. The underlying algorithms rely on the items' characteristics or the users' social contexts.

Elaine Rich kickstarted the field in 1979 with Grundy, the first-ever recommender system mainly designed to recommend books [1]. Her system asked users specific questions and assigned stereotypes to them using the responses. These stereotypes were then used to recommend books in a personalized way.

Launched in October 2006, the Netflix Prize was a major open competition that drove research in recommender systems. The challenge involved predicting user ratings on movies using collaborative filtering. The prize was awarded in 2009 to BellKor's Pragmatic Chaos team, which beat Netflix's algorithm by 10.06% [2]. These algorithms were very complex blends (hybrid) of other simpler methods. For instance, the best algorithm in 2007 used an ensemble method of 107 different algorithmic approaches blended into a single prediction [3].

A big challenge in recommender systems is the 'cold start' problem when new users or items are added. This issue arises in two ways: the 'new user' cold start, where the user has no prior data, and the 'new item' cold start, where there are no ratings for new items. We aim to study recommender systems and assess the performance of standard techniques (Collaborative filtering, content-based filtering, and hybrid methods) in tackling the cold start problem on the MovieLens dataset. We also combine these methods into hybrid models to improve performance. Our experiments offer insights into the relative performance of these techniques.

## 2 Algorithms

### 2.1 Collaborative Filtering (CF)

Collaborative filtering algorithms use ratings of many users to collaborate and recommend items to a user. The underlying assumptions are that:

- users with similar preferences on some items are likely to have similar preferences on others.
- users who have similar preferences in the past are likely to have similar preferences in the future.

To recommend items, the system collects data about the user in two ways:

- gather explicit ratings directly from users on a concrete scale (of five or ten, for e.g.).
- implicitly by using logs of the user in the system/platform or by other external means.

CF algorithms come in multiple flavors:

- Memory-based CF: This approach uses user-item interaction data to compute the similarity between users (used-based) or items (item-based). A subset of close neighbor users/items are selected that are similar to the target user/item. Similarity measures like Pearson correlation, cosine similarity, and Euclidean distance are used to compute the neighbors. These neighbors are then used to predict ratings for the target user/item, and the top recommendations are shown.
- Model-based CF: In this approach, statistical or machine learning models are developed to understand user preferences. Examples include Bayesian networks, clustering models, latent semantic models such as singular value decomposition, deep learning (autoencoders, RNNs, etc.), and latent Dirichlet allocation. Matrix Factorization methods involve decomposing the user-item interaction matrix into lower-dimensional matrices, aiding in the discovery of latent features underneath these interactions. Clustering models group users/items based on the interactions. These clusters are then used for recommendations.
- Hybrid Approaches: Hybrid approaches combine memory-based and model-based algorithms. Such approaches mitigate the shortcomings of each approach with the other and thus aim to produce a more robust model. However, they suffer from increased complexity and tend to be more expensive.

CF algorithms come with their unique challenges:

- Data Sparsity: The user-item matrix used in CF algorithms often tends to be very sparse since each user only rates a small portion of the big collection of movies. This issue is also reflected in the cold start problem when adding new users. These new users may not have other users similar to them based on the limited ratings by the new user. Similarly, new items can't be recommended until some users rate them.
- Scalability: As the number of customers reaches tens of millions and the number of items reaches millions, leading to an explosion in the user-item matrix size, an efficient CF algorithm with $O(n)$ complexity may already not be good enough.
- Synonymy: This refers to the inability of a CF algorithm to distinguish between items with different names that are essentially the same. For instance, 'children's movie' and 'children's film' are separate entities for a CF algorithm [4].

### 2.2 Matrix Factorization

Matrix factorization (MF) is a popular technique for building recommender systems, and singular value decomposition (SVD) is a specific MF algorithm with proven effectiveness. MF algorithms decompose a user-item interaction matrix into the product of two lower-dimensional matrices, capturing user preferences and item characteristics. This factorization represents users and items in a latent factor space, where factors represent underlying interests, genres, or attributes. By analyzing the latent factors, the recommender system predicts user ratings for unrated items, thereby recommending relevant items. [5]

- User Preferences: SVD identifies latent factors that represent user interests and preferences. For example, it might reveal that some users prefer historical novels while others enjoy science fiction.

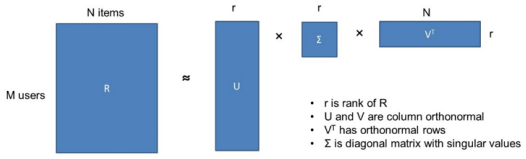


Figure 1: Singular Value Decomposition

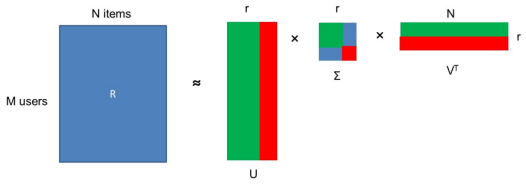


Figure 2: Compact representation of SVD

- Item Characteristics: SVD also identifies latent factors that represent item attributes and categories. These could be genre, author, publication date, etc.

The latent factors extracted through SVD are crucial for making recommendations. For example, a latent factor might represent a preference for action movies or a characteristic of having historical content. By analyzing the latent factors, the system can understand the relationships between users and items, even if they haven't interacted directly with each other. SVD breaks down the User-Item Interaction Matrix into three simpler matrices.

- User Matrix ($U$): Represents users in a lower-dimensional space, where each row corresponds to a user and each column represents a latent factor (e.g., interests, genres).
- Singular Values Matrix ($\Sigma$): Contains diagonal entries that indicate the importance of each latent factor.
- Item Matrix ($V$): Represents items in the same latent space as users, where each row corresponds to an item and each column represents a latent factor (e.g., attributes, categories).

Once SVD has extracted these insights, it can be used to recommend items to users. Given a user and an item they haven't rated, SVD can predict their potential rating. This is done by calculating the dot product of the corresponding user and item vectors in the latent space.

$$R = U \cdot \Sigma \cdot V^T$$

Based on the predicted rating, SVD can recommend items that the user is likely to enjoy. This allows for personalized recommendations beyond simply looking at what similar users have rated. However, there are challenges associated with SVD pertaining to Cold Start, Sparsity, and High Dimensionality.

### 2.3 Content Based Filtering

Content-based movie recommendation involves suggesting films to users based on the features and characteristics of movies they have liked or interacted with. The system analyzes movie features (e.g., genre, actors, director, popularity, budget, etc.) and user preferences to calculate similarity. Movies with features similar to a user's preferences are then ranked and recommended. Content-based filtering is constrained by the user's existing preferences, providing recommendations solely on what the user has liked.

#### 2.3.1 TF-IDF

It is a widely adopted technique in text mining and information retrieval, assesses the significance of words within a collection of documents. In this approach, TF (Term Frequency) is how often specific words occur in documents, with higher TF values indicating importance. DF (Document Frequency) reveals how frequently a word appears across the entire document collection; high DF values suggest

common words. The IDF (Inverse Document Frequency) is then used to measure word importance across all documents, with higher IDF values emphasizing rare words. TF-IDF is pivotal for keyword extraction, determining document similarity, and influencing search rankings, contributing to efficient information retrieval and document analysis [6].

- Term Frequency (TF) Calculation:
  For each document $d$ and keyword $k$, calculate the TF using the formula:

$$TF(k, d) = \frac{\text{Number of occurrences of } k \text{ in document } d}{\text{Total number of occurrences of words in document } d}$$

  Calculate TF for all keywords and documents.
- Document Frequency (DF) Calculation:
  For each keyword $k$, count the number of documents that contain $k$. Calculate the DF using the formula:

$$DF(k) = \frac{\text{Total number of documents}}{\text{Number of documents containing keyword } k}$$

  Calculate DF for all keywords.
- Inverse Document Frequency (IDF) Calculation:
  Compute IDF as the inverse of DF using the formula:

$$IDF(k) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing keyword } k}\right)$$

- TF-IDF Calculation:
  For each keyword $k$ in each document $d$, compute TF-IDF using the formula:

$$TFIDF(k, d) = TF(k, d) \times IDF(k)$$

  This step assigns a weight to each keyword in each document based on its frequency in the document and its rarity across all documents.
- Interpretation:
  Analyze the TF-IDF values to identify keywords that are important in each document, considering both their frequency in the document and rarity across the entire document collection.

#### 2.3.2 BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a state-of-the-art natural language processing (NLP) model developed by Google in 2018. [7] It is based on the Transformer architecture, which was introduced in [8]. Unlike traditional NLP models that read text sequentially (from left to right or right to left), BERT reads the entire input text bi-directionally. This bidirectional approach allows BERT to capture the context and dependencies between words more effectively, resulting in a deeper understanding of language semantics. The model consists of an encoder architecture with multiple layers of self-attention mechanisms. Here's a simplified explanation of the key components of BERT's model structure:

- Input Representation:
  BERT takes input in the form of tokenized text. The input is prepared by adding special tokens, such as [CLS] (classification) at the beginning and [SEP] (separator) between sentences or phrases. Each input token is represented as an embedding vector.
- Pre-training:
  BERT is pre-trained on a large corpus of text using two unsupervised tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). In MLM, random words in a sentence are masked, and the model is trained to predict the masked words based on the context provided by the other words in the sentence. In NSP, the model is trained to predict whether a sentence follows another sentence in a given text.

- Transformer Encoder:
  BERT utilizes the Transformer's encoder architecture, which consists of multiple layers of self-attention mechanisms and feedforward neural networks. Each encoder layer processes the input in parallel, allowing for efficient computation of contextualized word representations.
- Attention Mechanism:
  BERT employs self-attention mechanisms that enable each word to attend to all other words in the input sequence, capturing dependencies regardless of word order. The attention mechanism assigns different attention weights to different words, allowing the model to focus more on relevant information.
- Output Layer:
  The final hidden states of the [CLS] token in the last layer are used for downstream tasks, such as text classification or named entity recognition. Task-specific layers may be added on top of BERT's pre-trained layers for fine-tuning on specific applications.

BERT's ability to capture context, bidirectional information, and semantic relationships has made it a foundation for various NLP applications. The improved contextual understanding enhances accuracy by recommending items that align more closely with user preferences.

# 3 Dataset

We have executed our experiments on the *MovieLens-small* dataset containing 100,000 ratings. The dataset has 9,000 movies rated by 600 users. The dataset encompasses four key data files, namely *ratings.csv*, *movies.csv*, *tags.csv*, and *links.csv* [9]. While all files provide valuable insights, our primary focus rests on *ratings.csv*, *movies.csv* and *tags.csv* for the development and evaluation of our recommendation model. Key attributes in *ratings.csv* include *userId*, *movieId*, *rating*, and *timestamp*. These ratings, ranging from 1 to 5, illuminate user preferences and are the foundation for our algorithms. In *movies.csv*, each movie is identified by a unique *movieId* and includes information such as title and genre. The *tags.csv* contains user-applied movie tags, each entry representing a tag applied by one user to one movie.

# 4 Experiments

## 4.1 Simulating Cold Start

To simulate cold start conditions, we look at a user one at a time to recommend movies. We only use five movies rated by this user (the rest of the movies rated by this user are used for testing) and all the movies rated by all other users. This user acts as the new user in the system for which we intend to make good recommendations based on limited data on the user. An intersection of the movies recommended and the movies in the hold-out set is taken, and the mean squared error is calculated using the actual rating and the predicted rating on these movies over all the users. We then calculate the total error per user to compare different experiments.

## 4.2 User-based Collaborative Filtering

The algorithm is implemented in the following steps:

1. Identify the set $S_u$ of $k$-most similar users to the active user $u$. We use cosine similarity as the similarity metric to compare the similarity between users $u$ and $v$:
$$w_{u,v} = \frac{\sum_{i \in I} r_{u,i} r_{v,i}}{\sqrt{\sum_{i \in I} (r_{u,i})^2} \sqrt{\sum_{i \in I} (r_{v,i})^2}}$$
2. Find the most frequent items purchased by users in $G_u$ not purchased by $u$ yet.
3. Use weighted sum to aggregate ratings of these users on items found from the previous step. The predicted rating of user $u$ on item $i$ is:
$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in G_u} (r_{v,i} - \bar{r}_v) \cdot w_{u,v}}{\sum_{v \in G_u} |w_{u,v}|}$$

Here, the summation is over all users who have purchased item $i$. Subtracting the users' mean rating $\bar{r}_v$ compensates for differences in users' subjective sense of the rating scale, as some users give higher ratings than others.

4. Top $N$ items from the above step are finally recommended to the user $i$.

### 4.3 Matrix Factorization

User ratings are used to create the input matrix, with rows denoting userId, columns denoting movieId, and values representing ratings. First, the mean rating for each user is calculated and subtracted from these mean ratings to normalize the matrix. The normalized matrix is then factored into three matrices: $U$, $\Sigma$, and $V^T$ using SVD. Singular values can be found in the diagonal matrix $\Sigma$ [4]. The matrices $U$ and $V^T$, which represent users and movies, respectively, capture latent properties by using a reduced rank ($k = 2$). The predicted ratings are obtained by reconstructing the matrix from $U$, $\Sigma$, and $V^T$, and the mean ratings are added back. The code then evaluates the performance by calculating the mean squared error between the true ratings of unseen movies for each user and the predicted ratings.

### 4.4 Hybrid Recommender System-based Approach

Limitations of Singular Approaches:

- Content-Based Filtering may lack the ability to capture diverse user preferences.
- Collaborative Filtering may struggle when user-item interactions are sparse.

Our Hybrid Method combines the two methods to tackle the cold-start problem and improve the recommendation accuracy.

#### 4.4.1 Content-Based + Matrix Factorization

This model merges two recommendation approaches: content-based and matrix factorization. First, it suggests movies based on what users have liked before using content-based, understanding user preferences. Then, it fine-tunes these suggestions using collaborative filtering, which considers the preferences of people with similar tastes to the active user. The result is a list of recommended movies that balance the user's unique likes with broader trends among similar users, giving the user a personalized and well-rounded selection.

**Content Based (TF-IDF) Algorithm**

In the content-based approach, the algorithm analyzes the user's historical ratings and recommends movies that share similar content characteristics. This means that if a user enjoyed movies with certain genres in the past, the algorithm suggests new movies with comparable attributes. Genre serves as a significant meta-tag employed for categorizing movies with similar themes and characteristics. Predicts items by only looking at the user's information and doesn't consider input from other users, unlike collaborative techniques [10].

- Genre Vectorization: The first step involves converting textual genre information into a numerical format using a vectorization method called the TF-IDF vectorization technique, which transforms genre information into mathematical vectors to allow the system to represent genres as numerical features.
- Cosine Similarity Calculation: To determine the similarity between movies, the algorithm computes the cosine similarity using a linear kernel function. This measures the closeness of movies based on their genre vectors. The higher the cosine similarity score implies genres are more similar.
- User-Specific Recommendations: When a user requests recommendations, the algorithm considers their past movie ratings and identifies movies rated based on the genre information. The system then utilizes the similarity matrix obtained using cosine similarity to recommend movies that align with the genres the user has enjoyed in the past.
- Exclusion of Watched Movies: To enhance user experience, the algorithm ensures that recommended movies do not include those the user has already watched. This prevents redundant suggestions and encourages the exploration of new content.

**Matrix factorization Algorithm**

In the matrix factorization component, the algorithm analyzes user ratings and makes predictions based on collaborative patterns within the user community. By considering the preferences and behaviors of similar users, the algorithm refines its understanding of the user's tastes and predicts how they might rate other movies [11].

- Handling Missing Information: Matrix factorization introduces latent factors for both users and items. These factors capture hidden patterns or features in the data, allowing the model to make predictions even when ratings are missing.
  Predicted rating ($\hat{r}_{ui}$):
  $$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$
  Where $\mu$ is the overall average rating, $q$ is the vector associated with the item, $p$ is the vector associated with the user, and $b$ represents observed deviations/bias
- Minimizing Regularized Squared Error: The core objective is to minimize the regularized squared error using stochastic gradient descent. The algorithm adjusts biases (b) and factors (p and q) iteratively to make predictions closely aligned with actual ratings.
  Regularized squared error:
  $$\min_{q,p} \sum_{(u,i)\in\kappa} \left(r_{ui} - \mathbf{q}_i^T \mathbf{p}_u\right)^2 + \lambda \left(\|\mathbf{q}_i\|^2 + \|\mathbf{p}_u\|^2\right)$$
  where $\lambda$ is regularization parameter
  Stochastic Gradient Descent:
  $$e_{ui} = r_{ui} - q_i^T p_u$$
  $$q_i \leftarrow q_i + \gamma \left(e_{ui} \cdot p_u - \lambda \cdot q_i\right)$$
  $$p_u \leftarrow p_u + \gamma \left(e_{ui} \cdot q_i - \lambda \cdot p_u\right)$$
- Training Iterations: The algorithm iterates the entire training set for a predetermined number of epochs, 20 epochs in this instance.. During each epoch, the system optimizes user and item factors based on feedback from the training data, gradually refining its understanding of user preferences and movie characteristics.
- Personalized Recommendations: After the training process, the algorithm predicts how a user might rate movies, even for films they haven't seen. It recommends the top $N$ movies with the highest predicted ratings for the input user, tailoring suggestions to individual tastes.

### 4.4.2 Content-Based + $k$-NN

The hybrid method concurrently employs content-based and user-based collaborative filtering for movie recommendations. It begins by analyzing content similarity with highly-rated films by the user, providing recommendations based on this analysis. Simultaneously, it generates collaborative suggestions by identifying users with similar preferences through collaborative filtering. The concluding step involves aggregating recommendations and predicted ratings from both methods, utilizing predefined weights to achieve a well-balanced and comprehensive approach to movie recommendations.

**Content-Based (BERT) Algorithm**

A BERT-based content recommendation system utilizes pre-trained BERT embeddings to represent item features. By calculating similarity scores between rated movies by the user and movie embeddings, the system ranks movies and suggests the top-$N$ recommendations to users. This approach leverages BERT's semantic understanding to provide personalized recommendations, enhancing the user experience. The procedure below is described in [12].

- For every movie that user $u$ has rated, we identify the most similar movies by ranking the adjusted cosine similarity between the vectors generated by BERT. The Adjusted Cosine distance between two movies $i$ and $j$ is computed as follows:
  $$w_{u,v} = \frac{\sum_{u\in U}(r_{u,i} - \bar{r}_u)(r_{u,j} - \bar{r}_u)}{\sqrt{\sum_{u\in U}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{u\in U}(r_{u,j} - \bar{r}_u)^2}}$$
  This term is computed for all users $u \in U$, where is the set of users that rated both items $i$ and $j$.

- Identify the top 20 movies with the highest sum of similarity among the previously identified similar ones, excluding those rated by user $u$.
- Predict user $u$'s rating for movie $i$ by considering the rated movies by user $u$ and their respective similarities with movie $i$. The formula of predicted rating $\hat{r}_{u,i}$ for a given user $u$ on a movie $i$ is computed as follows:

$$\hat{r}_{u,i} = \frac{\sum_{j \in S_i} r_{u,j} \cdot w_{i,j}}{\sum_{j \in S_i} |w_{i,j}|}$$

  where $S_i$ is the set of top 20 similar movies to movie $i$.
- Recommend top N movies for user $u$ based on the predicted ratings.

**User-Based Collaborative Filtering Algorithm**

The method has been mentioned in the section 4.2.

# 5 Results

| **Recommendation Method** | **Loss per User** |
|---|---|
| Collaborative Filtering | 0.73 |
| Matrix Factorization | 3.63 |
| Content-Based + Matrix Factorization | 0.64 |
| Content-Based + $k$-NN | 0.69 |

Table 1: Loss per User for Different Recommendation Methods

Table 1 provides an overview of the evaluation results for various recommendation methods. Collaborative Filtering demonstrated effective prediction with a low loss, whereas Matrix Factorization exhibited a higher loss. The hybrid approach of Content-Based with Matrix Factorization showed a lower loss than just Matrix Factorization. Similarly, Content-Based with $k$-NN yielded a lower loss than just Collaborative Filtering. This shows that hybrid models indeed perform better than using just memory-based or model-based systems.

# 6 Conclusion

- Key insights:
  The experimental results reveal the trend of hybrid methods exhibiting lower losses in comparison to singular methods. This observation underscores the effectiveness of combining diverse approaches. Particularly noteworthy is the hybrid methods' ability to alleviate the cold-start problem, even in scenarios characterized by limited historical data. This suggests that the blending nature of hybrid models mitigates cold-start conditions, thus improving performance and leading to a more robust model.
- Challenges:
  The necessity to retrain the model for distinct cold-start conditions of each user contributes to computational complexity, yet this trade-off yields an improvement in accuracy. The challenge arises when integrating vastly different methods in hybrid models for prediction. Despite the trade-off between increased computational complexity and improved accuracy, these complex models can adapt to individual user conditions for optimal predictive performance.
- Future Work:
  For a more holistic analysis with bigger real-life datasets, we can use the Netflix Prize dataset or the bigger MovieLens datasets. If the model performs well in the new data environments, it will verify the robustness and adaptability of the model. Furthermore, we can use more complex hybrid methods and other sophisticated NLP algorithms to incorporate the genre and tag information in a better way. We can also use more advanced deep-learning techniques to tackle the problem and incorporate them into hybrid models.

## 7 Contributors

Every team member was involved in the project, ensuring a collaborative effort. Each member was responsible for one experiment. This included understanding the approach and coding it in Python. Nikhil worked on the User-based Collaborative Filtering approach. Felicia worked on the Matrix Factorization approach. Shreya worked on the Content-Based + Matrix Factorization hybrid approach. Tsu-Hao worked on the Content-Based + $k$-NN hybrid approach.

## References


[1] Rich, E. (1998). User Modeling via Stereotypes. *Cogn. Sci.*, 3, 329-354.

[2] Lohr, S. (2009, September 22). A $1 million research bargain for Netflix, and maybe a model for others. *The New York Times*.

[3] Bell, R., Koren, Y., & Volinsky, C. (2007). The BellKor solution to the Netflix Prize.

[4] Xiaoyuan Su, Taghi M. Khoshgoftaar, A survey of collaborative filtering techniques, *Advances in Artificial Intelligence archive*, 2009.

[5] Singular Value Decomposition in Recommender Systems `https://www.dataminingapps.com/2020/02/singular-value-decomposition-in-recommender-systems/`

[6] Kim, S. W., & Gil, J. M. (2019). Research paper classification systems based on TF-IDF and LDA schemes. Human-centric *Computing and Information Sciences*, 9, 30

[7] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2019)*, (Vol. 1, pp. 4171-4186).

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is All You Need. In *Proceedings of the Neural Information Processing Systems Conference (NeurIPS), 2017*.

[9] MovieLens [Dataset] `https://grouplens.org/datasets/movielens/`

[10] Pradeep, N., Rao Mangalore, K. K., Rajpal, B., Prasad, N., & Shastri, R. (2020) Content-Based Movie Recommendation System. *International journal of research in industrial engineering, 9(4), 337-348*

[11] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *IEEE.*

[12] Badrul Sarwar, George Karypis, George Karypis, Joseph Konstan & John Riedl. (2001). Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web* (pp. 285–295).

[13] Project Code (2023). cold-start. GitHub. `https://github.com/nik610/cold_start`