Learning Collective Embeddings for Item Cold-start Recommendations from Sentence Transformers

Farouq Oyebiyi

Masters by Research

University of York
Computer Science

December 2024

Abstract

Traditional Collaborative Filtering using Matrix Factorisation predicts user preference for every item in a catalogue, only accepting user and item identifiers as input. Without access to side information about the user and item, or adequate user-item interactions, it runs into a situation where it does not have enough information about a user or item to produce decent recommendations for either, a situation called cold start. One of the ways to alleviate this problem is to include side information i.e. metadata about the user or items. In this work, we propose to jointly factorise the partially-observed preference matrix and text-rich item side information. We represent the side information as a pairwise similarity matrix of embeddings derived from Sentence Transformers. This representation ensures we capture the relationships across every item. Our aim is to find item factors such that knowledge about the item metadata alleviates item cold start. To do that, we align the pairwise relationship between latent item factors from the decomposition of the user interaction matrix with the pairwise similarity matrix of sentence embedding by taking the Frobenius norm of their differences. This alignment penalty preserves global structure during joint factorization, capturing the semantic relationship across all items. The results obtained from our experiments show that our approach effectively exploits the rich semantic representation of text derived from Sentence Transformers to improve upon existing methods that relied on a sparse representation of side information and Collaborative Filtering.

Contents

List of Figures

List of Tables

Acknowledgements

My gratitude to Dr. Dimitar Kazakov for his guidance during this research.

Dedication

for Jumai, and Sameer

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, university. All sources are acknowledged as references.

Chapter 1

Introduction

Recommender systems are pervasive in our modern society. They have become a staple of everyday life, seamlessly integrating into many of our daily activities. One of the most noticeable areas where Recommender systems are prevalent is in the world of entertainment. Platforms such as Spotify, Netflix, and Youtube utilise these systems to suggest songs, movies, or videos based on our preferences and past behaviours, enhancing our overall user experience. However, the use of recommender systems extends far beyond entertainment. They have also made a significant impact on the domain of online shopping. Major e-commerce platforms like Amazon, eBay, and Shopify employ these systems to provide personalised product suggestions, thereby improving customer satisfaction and driving business growth. But the influence of recommender systems is not just limited to entertainment and shopping. These systems have started to make inroads into more critical environments. For instance, in the healthcare sector, recommender systems are being used to suggest treatment options based on a patient's medical history and current condition, potentially saving lives and improving healthcare outcomes. In the field of autonomous driving, recommender systems play a crucial role in suggesting the best routes and making real-time decisions, thereby enhancing safety and efficiency. The ubiquity of recommender systems in various aspects of our lives underscores their importance and potential for future applications.

Recommender systems are also employed as a method to reduce information overload. To truly comprehend the vast scale of recommender systems and the crucial importance of item cold-start, we must first take into account the immense size of a few popular catalogues we have identified are often part of everyday life. Consider

Pinterest[19], a platform that boasts over three billion pins, each one representing a different interest, idea, or product. Then there is YouTube which helps more than a billion users find personalised content from millions of videos [14]. It is a classic case of information overload, where the quantity of options can be more paralysing than liberating. This is why there is a necessity for a tool that aids in discovery, a tool that can analyse, sort, and recommend content based on the user's unique interests and preferences. Majority of items in such catalogues belong in the long tail i.e they have few interactions and as such not likely to be seen by users since popular items typically dominate interests.

Time spent looking for items that are of interest can be reduced by recommending to the user items the system thinks they are most likely to interact with. The system does this by considering the past interactions of the user and uses those to build a profile of what it thinks the user likes. User interactions on items play an important role in building Recommender Systems. These interactions are divided into two categories; implicit feedback, and explicit feedback. Implicit feedback is inferred by user actions on items. An example is a user reading an article - in which case, we can infer the user likes or is interested in content like that. Explicit feedback on the other hand are preferences given by a user to an item. An example in this case is a user giving an article a thumbs up - which means they like the article. These interactions, explicit and implicit, are useful in building a class of Recommender Systems known as Collaborative Filtering (CF). CF provides recommendations to users based on the similarity of their preferences to other users. This approach has the advantage that it can expand the user's taste by serendipity. Another advantage is that it does not require feature engineering or domain knowledge. Matrix Factorisation is a popular model for building Collaborative Filtering systems. At the core of the model is the factorisation of the user preferences into two compressed matrices such that the matrices capture the taste profile of the users and items. These matrices are also called latent matrices as they are thought to capture various characteristics of users and items [68]. The estimated preference for a user on an item is then the dot product of the user latent factors and the item latent factors.

Another approach to building Recommender Systems is called Content-based Filtering. This approach uses the similarity of metadata about the user or item to make recommendations. Methods for building Content-based Filtering often relies on domain knowledge and engineering features for users and items to accurately match user interest. [67]

For recommender systems to be effective, it is not sufficient to only predict the preference for each user over all items. They need to rank the preferences – presenting to the user a short ranked list of items the user is most likely to interact with. This is top-N recommendation, where N is a number decided by the domain. This ranking is crucial to capture the attention of users.

One major failing of Collaborative Filtering is that it does not consider metadata about items, or users, relying solely on the interactions of users on items. Often, this user-item interaction is sparse, leading to cold start[50]. This can take the form of a new user who has not watched any movies or a new or obscure item that has not been considered by enough users. The challenge then is how to provide useful recommendations to this user who has little or no consumption history.

Our work is an extension of the research conducted by Singh [63], which introduced the concept of learning collectively from multiple relations via the technique of Collective Matrix Factorisation (CMF). The decompositions in this technique are primarily composed of shared latent factors. This is because the core assumption of this approach is that there exists a shared subspace among all entities involved in the relations. This shared subspace serves as a common ground that binds these entities together, aiding in the collective learning process. The way this approach addresses the cold start problem, which is a common issue in recommendation systems when new users or items that have no rating history is by incorporating metadata about the user and the item into the factorisation process. This metadata could include information about the user's preferences or details about the item, which enriches the factorisation process and allows for better handling of new, unrated items or users, thereby mitigating the effects of the cold start problem. In this work, we posit that the cluster hypothesis introduced by Rijsbergen et al. [58] can be combined with traditional Matrix Factorisation to alleviate the item cold-start problem. We propose a single stage model which jointly factorises the user preference matrix and the side information matrix. In order to factorise the side information matrix, we employ Symmetric Matrix Factorisation (SMF), which has been shown to be an approximation of K-Means clustering [76].

1.1 Contributions

In this thesis, we extend CMF by using Symmetric Matrix Factorisation for the decompositon of the item side information which is represented using a pairwise similarity matrix. In addition, we investigated the effect of using embeddings from sentence transformers as features for the side information, as opposed to binary features or TF-IDF which is routinely used.

Chapter 2

Background and Related Work

2.1 Recommender Systems

2.1.1 Components

Before diving into the details of Recommender Systems, we first describe the major components of such a system.

User Profiles

Recommendations are provided to users, as such, they are a critical component. User profile contains information which represents the identity and preferences of a user. These identity information can be collected such as when the user signs up to the system and can include various types of data such as demographic data showing the age, location and gender. Identity data can also include information about the devices, IP addresses and browsers the user uses to interact with the system. These identity and preference data about the user is also called user metadata. Contextual information about users such as devices, time of day or location are often dynamic and a good recommender system needs to be able to adapt to changes. Ideally, the more metadata we have about a user, the better we can capture their preferences and provide them with relevant recommendations. However, privacy concerns often limits the scope of the data one can capture. User metadata is good for content-based filtering - an approach to building recommender systems which we will be looking into in the next section.

Item Profiles

While a user profile constitutes data describing a user, item profile consists of data that describes an item. Item metadata depends on the domain, and thus varies widely. Examples of item metadata for a video streaming platform such as Netflix would include title, description, running length, cast, director, location, genre etc. While for an e-commerce platform such as Amazon, we can expect to have item metadata such as category, price, variant, colour, size, quantity etc.

As with user metadata, we also encounter the issue of dynamic data where metadata is not static and can change quite often. An example is the price of an item on an e-commerce store, or the presence of a discount. In addition, some item metadata are unstructured text, which then requires sophisticated techniques such as Natural Language Processing to extract the relevant features. Consider the plot or synopsis for a movie for example. Usually, this contains the theme, various scenes, characters and story all laid in free-form.

As with user metadata, item metadata is essential for content-based filtering, as it allows a recommender system to find items that have similar metadata.

Feedback

Now, we turn to feedback, which encompasses user interactions with items in the system. Such user behaviour includes browsing history, purchases, clicks, views, listening history etc. This behaviour is domain specific and so varies. An ecommerce platform, say, Amazon would include behaviour data such as purchase history, wish list history, product rating. While a music streaming platform e.g. Spotify would include behaviour data, such as listening history, liked playlists, liked songs e.t.c.

These interactions are crucial as feedback as they signal to us what the user preferences are. We can divide this feedback into two: implicit, and explicit.

Explicit feedback are those provided by the users. It can be ordinal on a scale, say, from 1 - 5, in increasing order of satisfaction. Or it can be binary, such as a like/dislike, thumbs up/thumbs down. And it can also be free text, in the case of product or movie review - where the user leaves a comment about what they think of the item. With explicit feedback, the user can be quite clear to what extent they liked an item.

One problem with explicit feedback is that it requires active participation on the part of the user. They have to care enough about the item to want to provide

feedback, or be prodded, often annoyingly, to leave one. The consequence of which then is the sparsity of explicit feedback.

Another problem that arises with explicit feedback is the choice of feedback mechanism. A rating scale such as a 5-star might suffer from central tendency bias, where users avoid extremes and hence choose a rating in the middle, while binary feedback does not capture nuanced preferences.

Implicit feedback on the other hand is inferred from the interactions of the user in the system. For example, watching an entire movie might indicate an interest in that genre, or sequel of the same title. User behaviour such as views, clicks, purchase history etc. is used to make a judgement on users' interest.

What this means is there is a wealth of data for implicit feedback since user interactions are natural, and hence abundant. However, such feedback is noisy and difficult to interpret. Consider the case where you bought an item for a family member off your Amazon account as a one-off - an item which the account owner has no interest in. That interaction could possibly be used to enrich, rather wrongly, the owner's account.

2.1.2 Similarity Measures

The user and item profiles are usually converted to a N-dimensional vector for use by the recommender system. A crucial component of the system is then a notion of similarity to measure the closeness of user vectors and item vectors. This is useful, because if we know a user is interested in a movie, say, *Dr. No*, then, we can find movies whose vectors are similar to this query vector and recommend them to the user.

Jaccard Coefficient

The Jaccard Similarity, which is also known as the Jaccard Index or Jaccard Coefficient, measures the similarity between sets. It is defined as the size of the intersection divided by the size of the union of two sets. The formula is presented below:

$$j(a, b) = \frac{|a \cap b|}{|a \cup b|} \tag{2.1}$$

where $|a \cap b|$ is the number of elements in the intersection of a and b, and $|a \cup b|$ is the number of elements in the union of a and b. Similarity ranges from 0 to 1.

Similarity is 0 there is not overlap between the sets, and it is 1 if there is a perfect overlap.

Jaccard similarity has the downside that it only considers interactions defined as sets. Below, we introduce a few similarity techniques without this shortcoming.

Cosine Similarity

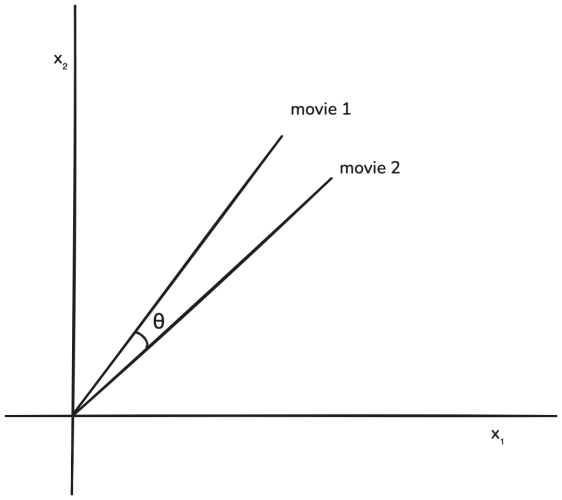

Figure 2.1: Cosine Distance between two movies

Cosine Similarity is one of the most commonly used methods for measuring the similarity of real-valued vectors. It is used in Information Retrieval to measure the similarity of documents. Cosine Similarity calculates the cosine of the angle between two vectors which is obtained by dividing the inner product of the vectors by the product of their Euclidean norm. The angle θ measures to what extent the two vectors point in the same direction, irrespective of the magnitude as the two vectors are normalised. The formula is given below:

$$cos\theta = \frac{< a, b >}{||a||.||b||} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2.2}$$

a, b are real-valued vectors and $< . >$ is the inner product of two vectors. The values for Cosine Similarity lie in the range -1 to 1. 1 indicates perfect similarity,

0 indicates orthogonality, i.e the vectors are at right angles to each other. And -1 means the vectors are diametrically opposed.

Cosine Similarity is appropriate in instances when dealing with sparse vectors as well as high dimensions. Another useful property of Cosine Similarity is that it measures orientation, not magnitude. Thus, two vectors can have high similarity with different magnitudes.

Pearson Coefficient

The Pearson Correlation Coefficient, often denoted as 'r', measures the linear correlation between two variables. In the context of recommender systems, it measures the strength and direction of the linear relationship between user interaction data such as ratings and other kinds of feedback, and item features. The formula is given below:

$$r(a,b) = \frac{\sum_{i=1}^{|a|}(a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{|a|}(a_i - \bar{a})^2}\sqrt{\sum_{i=1}^{|b|}(b_i - \bar{b})^2}} \tag{2.3}$$

a, b are real-valued vectors and $|.|$ is the number of elements in the vector. The values for Pearson Coefficient lie in the range -1 to 1. 1 indicates perfect linear correlation, 0 indicates no correlation, and -1 means the vectors have a perfect negative linear correlation. The formulation above is quite similar to the definition of Cosine similarity, with the only difference being that for each value in the vector, we subtract the mean of the corresponding vector.

One advantage of the Pearson Coefficient over the previous similarities we looked at is that it is able to capture linear relationships between variables while adjusting for individual rating scales of the users. For example, consider two users, $user_1$ and $user_2$, who rate two movies $movie_1$, and $movie_2$, as [3, 5] and [5, 3] respectively. Our definition of Cosine similarity and Jaccard similarity would find these users to be similar based on their ratings. However, we can see from the rating vectors that these users are polar opposites. From the formula provided, we see Pearson Correlation adjusts for the mean rating of each user, making it more suitable when users have different baseline rating tendencies.

2.1.3 Content-based Filtering

Content-based filtering in recommender systems uses the metadata of users and items in the system to provide personalised recommendations to users [43, 3, 2, 41]. Using interaction data of users, recommendations then becomes a matter of finding items similar to the ones the user has interacted with. For example, if a user's interaction history on a movie platform consists of *Harry Potter and the Philosopher's Stone*, and *Harry Potter and the Chamber of Secrets*. The user might get *Harry Potter and the Prisoner of Azkaban* as a recommendation since this is another title in the series. Or they might be recommended *The Chronicles of Narnia: The Lion, the Witch and the Wardrobe*, which is a title in the genre Fantasy, as the *Harry Potter* series.

Content-based filtering is straightforward and easy to implement, making it ubiquitous in recommender systems and is often the first step when adopting personalised recommendations. It has several useful properties that make adoption easy.

One such property is that content-based filtering does not require a lot of interaction data to provide recommendations. This is especially useful when the platform is new and user interaction data does not exist or is limited. However, by using user and item metadata the system can provide recommendations to the users.

Another useful property is that content-based filtering is able to recommend items in the long tail, those items which are new to the system or have little to no user interaction. This property of content-based filtering is crucial for platforms which have a frequently updated catalogue or inventory. As new items are introduced into the platform, they run the risk of not being discovered by users as they will not have any interaction data. But this is mitigated by the presence of metadata which is then used to match the items with appropriate users.

Content-based filtering is intuitive and highly explainable. It is easy to understand the rationale behind the recommendations provided. For example a recommendation to watch *Harry Potter and the Prisoner of Azkaban* can be easily explained by "Because you watched *Harry Potter and the Philosopher's Stone*". This transparency fosters user trust and provides valuable insights. This intuitive approach to recommendations is also natural to how humans think. All this contributes to its ready adoption.

In addition, content-based filtering relies on preferences explicitly provided by the user in terms of metadata, and as such is less prone to manipulation when compared to approaches relying on frequently changing data such as user ratings or interactions.

While we acknowledge the various useful properties of content-based filtering, it is not without some disadvantages. One major disadvantage is the reliance on hand-engineered metadata, which is also domain-dependent. Each platform needs to decide beforehand which kinds of attributes they require from the user and how these interact with the item metadata. This task is time consuming and is a source of friction as it requires participation on the part of the user.

Another notable disadvantage is overspecialisation. Content-based filtering is only able to recommend items similar to what the user liked or interacted with, it can trap the user in what is known as a filter bubble. This term refers to the lack of diversity in the recommendations provided to a user as the system is incapable of offering them content outside their stated interests. Similar to overspecialisation is the lack of serendipity. Serendipity is the accidental discovery of items which are of interest to a user but were not sought after. Overspecialisation in content-based filtering would result in more of the same kind of recommendations, boring the user. However, with pleasant discoveries of new or adjacent items, the user's taste is expanded and therefore engaged.

2.1.4 Collaborative Filtering

Collaborative filtering relies on collective user behaviour to provide personalised recommendations [11, 33, 59, 5, 8]. The idea behind collaborative filtering is that users who have similar interactions and preferences for items also have similar tastes. Hence, personalised recommendation for a user rely on finding users with similar interests and filtering for items this user has not interacted with based on the preferences of similar users. This runs contrary to the approach we saw earlier, content-based filtering, which relies solely on user and item metadata. Collaborative filtering can be user-based, where similarities are computed between users, or item-based, where the similarities is between items. Collaborative filtering requires historical interaction data to compute this similarities. Interaction data varies and is platform dependent. For example, on Netflix, a video streaming platform, interaction data for Collaborative Filtering could be user's streaming history and watch list. For Pinterest, a social media platform, interaction data suitable for collaborative filtering could be the Pins added by users, their followers etc.

Unlike content-based filtering, which requires domain-specific metadata to be engineered, collaborative filtering is domain independent. This property makes it

valuable as it can be easily deployed without having extensive knowledge of the properties of the target domain.

Another useful property of collaborative filtering is how it helps with discovery via serendipitous recommendations. Since the core of collaborative filtering is finding patterns in preferences of similar users, the recommendation for a user can be drawn from a diverse pool of items, expanding the user's taste while keeping them engaged.

As the quantity of interaction data grows, the quality of recommendations provided by collaborative filtering also increases. User interactions with the platform provide valuable feedback which helps collaborative filtering algorithms better capture nuanced and subtle patterns of behaviour that are difficult to capture by metadata provided by content-based filtering systems.

Next, we turn to some of the challenges with collaborative filtering. Perhaps the most significant challenge is the cold start problem. It is difficult to provide accurate personalised recommendations to new users since they have no interaction history. Without interaction history, it is impossible to find similar users which are needed to predict the recommendations for the new user. This also affects new items - those with no interaction data provide no signal for use in collaborative filtering.

A corollary to above is the data sparsity. For new systems or ones with a very extensive catalogue of items and users, interaction data is often sparse. This sparsity affects the quality of the personalised recommendations a collaborative filtering algorithm is able to produce. Since users would only have interacted with a small number of items in a large catalogue, there would not be enough signal in the data to capture the preferences for users.

Collaborative filtering systems can be prone to popularity bias, where popular items are recommended more frequently while niche items are overlooked. This can lead to a rich-get-richer effect, potentially limiting the diversity of recommendations.

Another challenge inherent in collaborative filtering algorithms is popularity bias. As noted by [1], recommendations provided by collaborative filtering approaches are often dominated by popular items, to the detriment of items belonging in the long-tail.

Lastly, compared to content-based filtering, explanations for recommendations provided by collaborative filtering, especially those from model-based approaches, are less intuitive.

Collaborative Filtering is divided into two distinct methods based on how recommendations are computed. First, we have a memory-based approach, it computes

recommendations by taking the similarities between users, and or items over the entire interaction data. It is also called the neighbourhood method. While easy to implement, this approach is not scalable as the inventory and interaction data grows, as the computation requires going over the entirety of the data. The other approach to collaborative filtering is called model-based, it involves using techniques like Machine Learning or Data Mining to discover patterns in the interaction data which are then used to make personalised recommendations by predicting the users' preference for items they have not interacted with. Memory-based approaches are better able to capture nuanced behaviour and provide quality recommendations. In addition, they scale very well with increasing data, compared to model-based approaches, since the resulting model is often orders of magnitude smaller than the dataset. Another advantage of model-based approaches is that they can handle the sparsity of user interaction data much better than memory-based approaches. An important algorithm for building model-based collaborative filtering is called Matrix Factorisation. It is at the heart of this work and we will be exploring it in detail in the next section.

2.1.5 Matrix Factorisation

Matrix Factorisation [38] is a class of collaborative filtering algorithms that gained popularity after the Netflix Prize held in 2006. This contest was announced by Netflix to see if anyone could build a recommender system that would beat a baseline established by Cinematch by 10%. The reward for the competition was $1 million. Matrix Factorisation gained widespread attraction after the report by Simon Funk in his blog post which explored the effectiveness of the method. Like all model-based approaches, it uses Machine Learning to predict the preferences of users from user interaction data. The core idea behind Matrix Factorisation is to decompose the user interaction matrix into the product of two lower rank matrices which are in the same embedding space. This factorisation captures the lower-rank structure of the user interactions, which is assumed by this class of models to summarise user behaviour.

Model

The user preference matrix, R, is factorised into two matrices, P, Q which are latent feature matrices for users and items respectively, depicted in Figure 2.2. k denotes

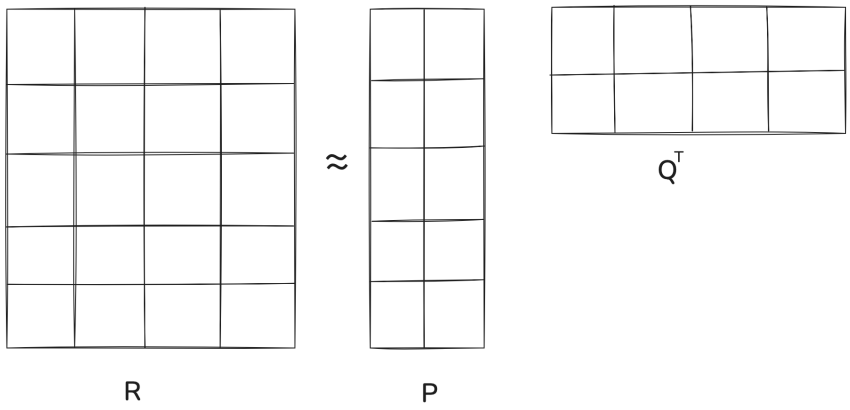

Figure 2.2: Matrix Factorisation

the rank of the latent feature matrices. This projection into a lower-dimensional space has the effect of detecting features that capture the structure of R as well as approximates it - this approximation we denote $\hat{\mathrm{R}}$.

$$R \approx \hat{R} = PQ^T = \begin{bmatrix} p_1^T \\ p_2^T \\ p_3^T \\ . \\ . \\ . \\ p_N^T \end{bmatrix}_{N \times k} \begin{bmatrix} q_1 & q_2 & q_3 & . & . & . & q_M \end{bmatrix}_{k \times M} \tag{2.4}$$

User and Item biases

From the prediction rule defined in equation 2.4 above, we see that the model does not consider user and item biases. We can modify the model by including bias terms for users and items. The justification for these is that users often differ in how they experience an item in the catalogue and the resulting rating or feedback they provide. For example, some users will tend to provide only negative feedback, while others will only rate items highly. Conversely, popular items often receive a lot of ratings while items in the long tail scarcely so. These biases captures user behaviour. We modify our initial model as to include bias terms:

$$\hat{R} = R - PQ^T + \beta^u + \beta^i \tag{2.5}$$

where β^u and β^i are respectively user and item bias.

Regularisation

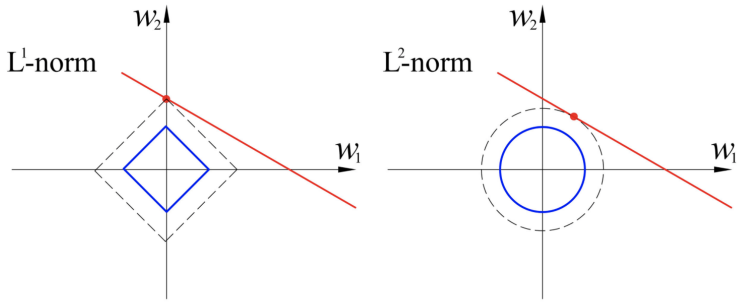

Figure 2.3: L1 and L2 norm showing sparsity and uniform distribution respectively

The parameters in a model are sources of complexity. Too many parameters and the model overfits the data because it is too complex and fits the training data too well. Such a complex model will not generalise to unseen data and hence perform poorly. What we want is for our model to fit the training data well enough but also to generalise. To control overfitting, instead of reducing the number of parameters in our model, we introduce penalty terms into our objective function. This has the goal of restricting the range of values the parameter can take. We look at two approaches for capturing the complexity of the model based on their norm of the parameter. The norm,

$$||.||$$

is a mathematical function that tells us how big a matrix of vector is.

L_1 Norm: Given a parameter vector, θ, the norm of the vector is the sum of absolute values in θ.

This is given by:

$$\Omega_1(\theta) = ||\theta||_1 = \Sigma_k|\theta_k| \tag{2.6}$$

The L_1 norm has a useful property where it has a high penalty for models with many nonzero parameters. This typically encourages a sparse parameter vector which is often useful for feature selection.

L_2 Norm: Given a parameter vector, θ, the norm of the vector is the sum of squared values in θ.

This is given by:

$$\Omega_2(\theta) = ||\theta||_2^2 = \Sigma_k \theta_k^2 \tag{2.7}$$

L_2 norm on the other hand places a large penalty on large parameters resulting in a model with a more balanced parameter distribution which makes the model robust to outliers.

While the norms defined above are specific to vectors, a useful norm for matrices, called Frobenius norm is defined below:

$$\Omega_F(\theta) = ||\theta||_F^2 = \Sigma_j \Sigma_k \theta_{jk}^2 \tag{2.8}$$

Frobenius norm is an extension of the L_2 norm to matrices. And this is useful since several of the parameters in our Matrix Factorisation model are matrices.

Now, we can modify our objective to include regularisation using Frobenius norm like so:

$$\min_{P,Q}[||R - PQ^T||_F^2 + \lambda(||P||_F^2 + ||Q||_F^2)] \tag{2.9}$$

Loss Functions

Finding the model parameters that best describe the user interaction then results in minimising an appropriate objective. For explicit user interactions which are ordinal such as a rating, we want to minimise the error between the actual rating and the predicted rating. We consider two such approaches below.

Root Mean Squared Error

This is the square root of the sum of squared differences between the predicted ratings and the actual ratings. Root Mean Squared Error (RMSE) is defined as follows:

$$RMSE = \sqrt{\frac{1}{|\hat{R}|}\Sigma_{\hat{r}_{ui}\epsilon\hat{R}}(r_{ui} - \hat{r_{ui}})^2} \tag{2.10}$$

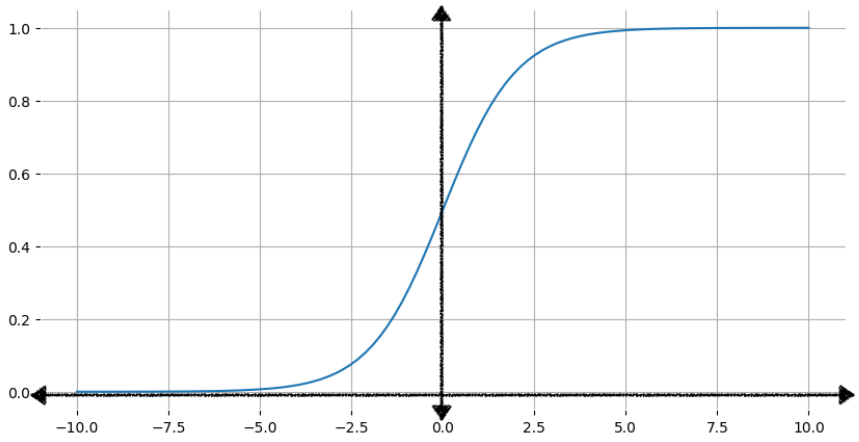

Figure 2.4: Sigmoid Function

Mean Absolute Error

This is the sum of the absolute values of the differences between the predicted ratings and the actual ratings. Mean Absolute Error (MAE) is defined as follows:

$$MAE = \frac{1}{|\hat{R}|}\Sigma_{\hat{r}_{ui}\epsilon\hat{R}}|r_{ui} - \hat{r_{ui}}| \tag{2.11}$$

In both instances above, we take the average of the loss over the entire predictions. MAE and RMSE, are both easy to use and simple to explain. They do however differ in an important way. RMSE, due to its squared term penalises large errors severely. This results in the model being affected by outliers or bad predictions. MAE on the other hand weighs the errors equally and is not sensitive to large errors.

The choice of loss function depends on whether one favours having a comprehensive view of rating accuracy or being sensitive to outliers.

$$\min_{P,Q} ||R - PQ^T||_F^2 \tag{2.12}$$

However, in the case where our user interaction matrix consists of implicit feedback, that is, user preference is represented by binary values, the loss functions defined above are not sufficient to capture user preference.

Log Loss

With Log Loss, user preference prediction is treated as a binary classification problem, where 1 indicates preference for the item. The task can then be formulated as predicting the probability of the positive class. To that end, we use the sigmoid function which maps all real numbers on to the (0, 1) interval as shown in Figure 2.4. We see that as the value tend to higher positive values, the sigmoid function tends to one, and goes to 0 as the input tends to $-\infty$. The formula for sigmoid is given below.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

With the sigmoid function defined above, the loss function for our implicit dataset is then given below:

$$LogLoss = \frac{1}{|\hat{R}|}(\Sigma(-r_{ui} log(\sigma(\hat{r_{ui}})) + (1 - r_{ui}) log(1 - \sigma(\hat{r_{ui}})))) \tag{2.14}$$

Optimisation Algorithms

We define an objective function for a model using log loss as the loss function, with bias and regularisation terms below.

Finding the parameters of the model that best fits the data is often referred to as fitting the model. For our case we want to minimise the log loss of the predicted preference of the user on an item. Lower values indicate smaller error in prediction, with higher values being the converse. To fit a model to data, there are a handful of approaches but we take a look at two popular approaches to solving Matrix Factorisation problems.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an iterative optimisation algorithm used to optimise the parameters of a model. The gradient evaluated at a point gives the direction and rate of steepest increase of the function. SGD updates the parameters of the model iteratively by taking a step in the opposite direction of the gradient - as this is the steepest descent. Unlike Gradient Descent which computes the gradient using the entire dataset, SGD estimates the gradient using a randomly selected sample at

each time step. The frequent update performed by SGD causes the loss to fluctuate heavily.

Algorithm 1 Stochastic Gradient Descent for Matrix Factorisation

procedure SGD(R, k, η, λ, n)
    Input: R: User-item interaction matrix. k: Number of latent factors. λ: Regularization parameter. n: Number of iterations. η: Learning rate
    Initialize $P \in \mathbb{R}^{m \times k}$ and $Q \in \mathbb{R}^{n \times k}$ with small random values
    for $iter = 1$ to n **do**
        for each (u, i, r_{ui}) in R **do**
            $e_{ui} \leftarrow r_{ui} - p_u^T q_i$
            $p_u \leftarrow p_u + \eta(e_{ui} q_i - \lambda p_u)$
            $q_i \leftarrow q_i + \eta(e_{ui} p_u - \lambda q_i)$
        end for
    end for
    Return P, Q
end procedure

Rather than using only one sample to estimate the gradient, mini-batch gradient descent uses a batch of samples. Using mini batches to estimate the gradient reduces the variance of parameter updates inherent in SGD. This also benefits from vectorised matrix computations for efficient computation. Update to SGD such as Momentum [54] help to move SGD quickly in the relevant direction by dampening oscillations. In addition, so called Adaptive methods such as Adam[37], Adagrad[18] and Adadelta[75] have been developed with the idea being to adapt the learning rate to the frequency of updates of each parameter. This is achieved by keeping a history of past gradients.

Alternating Least Squares

This is also an iterative optimisation algorithm. It updates the parameters of the model by taking turns to optimise one parameter while keeping the rest fixed. It continues this process until convergence.

Algorithm 2 Alternating Least Squares for Matrix Factorisation

procedure ALS FOR MATRIX FACTORISATION(R, k, λ, n)
  Input: R: User-item interaction matrix. k: Number of latent factors. λ: Regularization parameter. n: Number of iterations.
  Initialize $P \in \mathbb{R}^{m \times k}$ and $Q \in \mathbb{R}^{n \times k}$ with small random values
  for $iter = 1$ to n **do**
    for $u = 1$ to m **do**
      $p_u \leftarrow (Q_{I_u}^T Q_{I_u} + \lambda I_k)^{-1} Q_{I_u}^T R_{u,I_u}$
    end for
    for $i = 1$ to n **do**
      $q_i \leftarrow (P_{U_i}^T P_{U_i} + \lambda I_k)^{-1} P_{U_i}^T R_{U_i,i}$
    end for
  end for
  Return P, Q
end procedure

Relationship to Singular Value Decomposition (SVD)

Matrix Factorisation (MF) as described above is closely related to SVD. MF is a generalisation of the decomposition of a matrix into a product of two or more matrices, while SVD is specific in that the matrix is decomposed into a product of three matrices. With SVD, the decomposition is given as:

$$M = U \Sigma V^T \tag{2.15}$$

Matrices U and V are orthogonal matrices whose columns and rows are unit length vectors. The matrix Σ on the other hand is a diagonal matrix containing singular values in its diagonal. The singular values are ordered from largest to smallest. The geometric intuition for SVD is that the decomposition of a matrix first goes through a change of basis via the V matrix, and then a scaling via the singular values in Σ, and then another change of basis via the U matrix. Unlike SVD, MF described above does not place any constraints on the properties of the matrices that make up the decomposition.

2.2 Word Representation in NLP

For Machine Learning Algorithms to make sense of text, they need to be converted to a numerical format. We examine a few Natural Language Processing (NLP) techniques available to convert raw text into numerical format. We look at basic statistical techniques which use frequency counts to represent words and sentences, and proceed to state-of-the-art approaches capable of providing semantic representation of whole sentences.

2.2.1 Term Frequency-Inverse Document Frequency

One such approach is Term Frequency - Inverse Document Frequency (TF-IDF) [64] [61]. This is a classical technique in Information Retrieval used to weight the importance of words by the frequency of its occurrence in the document collection. It considers two factors:

1. Term Frequency (TF), which determines how frequently a term occurs in a specific document. A higher TF in a document often means that term is important.

2. Inverse Document Frequency (IDF) on the other hand denotes how rare a term is across the entire document collection. Words that appear frequently across many documents are less informative, so their IDF score is lower.

By combining these factors, TF-IDF assigns higher weights to words that are frequent within a specific document but rare overall. These words are likely to be more informative and capture the document's unique theme. However, one shortcoming of TF-IDF is that it does not consider the context of the term, and as such does not capture the semantics of words. In addition, it is sensitive to frequently occurring words, known as stopwords, requiring domain expertise to filter these out.

2.2.2 Word2Vec

Distributed word representation, also known as word embeddings, on the other hand address these shortcomings. They embed words as vectors in a high-dimensional space, where words with similar meanings are positioned closer together in the space.

These embeddings capture semantic relationships between words based on their co-occurrence patterns in large text corpora. Word2Vec[42], GloVe[52] and sentence transformers[55] are techniques for learning word embeddings.

2.2.3 Transformer Architecture

The Transformer architecture was proposed in [69], and since its introduction, it has led to significant improvements across fields outside Natural Language Processing (NLP). Figure 2.5 shows the main components of the Transformer architecture. Initially developed to improve Machine Translation tasks, Transformers have found widespread use in various domains of artificial intelligence and machine learning. In Computer Vision, Transformer-based models have shown remarkable performance in tasks such as image classification, object detection, and image segmentation. In Reinforcement Learning, Transformers have been applied to enhance the performance of agents in complex environments. Their capacity to process sequential data and maintain context has proven beneficial in decision-making tasks. Recommender Systems have also benefited from Transformer architectures specifically due to their suitability for processing sequential data as well as better representation learning. This has led to various approaches to sequence-aware models using Transformers [35, 66, 40].

The Transformer architecture has become a main component of Large Language Models (LLMs) - language models with billions of parameters capable of diverse tasks. These LLMs can perform a wide range of language-related tasks, including:

1. Comprehension: Understanding and interpreting complex text passages;
2. Summarization: Generating concise summaries of longer texts;
3. Translation: Converting text from one language to another with high accuracy;
4. Keyword extraction: Identifying and extracting key terms or phrases from text;
5. Conversation: Engaging in human-like dialogue across various topics.

The effectiveness of Transformer-based models have made them a cornerstone of modern AI research and applications, continuing to drive advancements in multiple fields of study and practical applications.

Prior to the development of the Transformer, Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks were the dominant approaches to modeling sequence tasks in Natural Language Processing. RNNs operate on se-

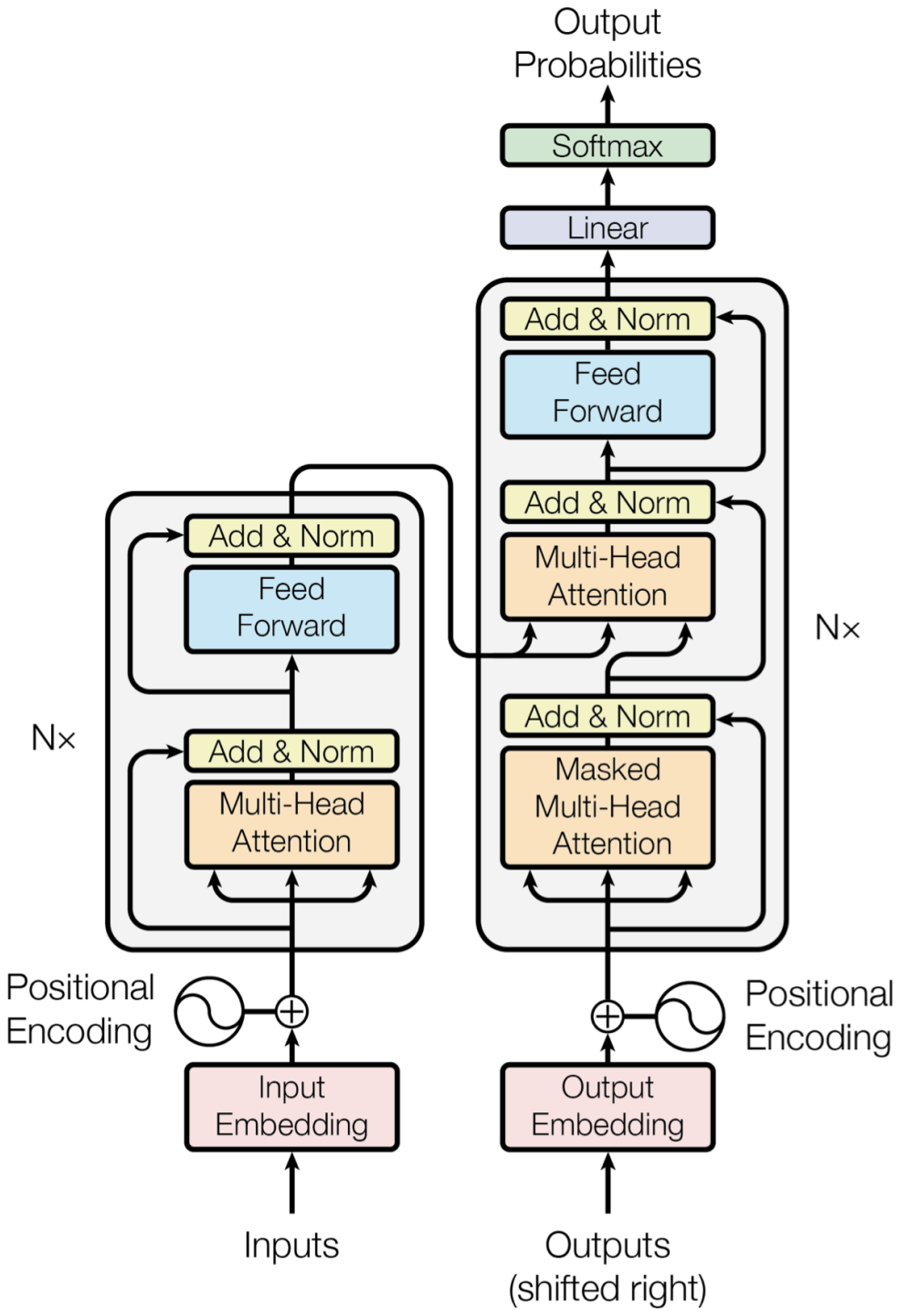

Figure 2.5: Transformer Architecture [69]

quences such that there is a hidden state for each of the tokens in the sequence. This state contains information about all the tokens in the sequence up to the current token. Training RNNs, however, runs into the vanishing gradient problem. This was first identified by Hochreiter in his diploma thesis and later published in 1998 [32]. The vanishing gradient problem occurs when gradients diminish as they are propagated back through the network, thus making weight updates difficult and learning effectively impossible for long sequences.

A successor to RNN is the LSTM [31], which tackled the vanishing gradient problem present in the former. LSTMs introduced memory cells which store information and gates which controlled the flow of information from previous hidden states onto subsequent ones. These gates include:

1. Input gate: Controls what new information is stored in the cell state
2. Forget gate: Decides what information should be discarded from the cell state
3. Output gate: Determines what information from the cell state should be used as output

While LSTMs improved upon RNNs, they still faced limitations in processing very long sequences and capturing long-range dependencies effectively. Additionally, both RNNs and LSTMs process sequences sequentially, which limits their parallelisation capabilities and makes them computationally expensive for long sequences. These limitations of RNNs and LSTMs set the stage for the development of the Transformer architecture, which addressed many of these issues through its novel attention mechanism and parallel processing capabilities.

We turn to the various components which make up the Transformer architecture.

Self-Attention

Self-Attention is a mechanism that encodes a token in the sequence making sure to include context about all the other tokens. Self-Attention introduces queries, keys and values, which are linear transformations of each token in the sequence by three respective weight matrices. Each token is transformed into a query and is compared against the keys for every other token in the sequence, producing what is called an attention score over the tokens in the sequence. The representation for any token is then the weighted sum of the product of the attention and token values. This is the scaled dot-product attention introduced in 2017 [69]. This mechanism allows the model to focus on different parts of the input sequence when encoding each token,

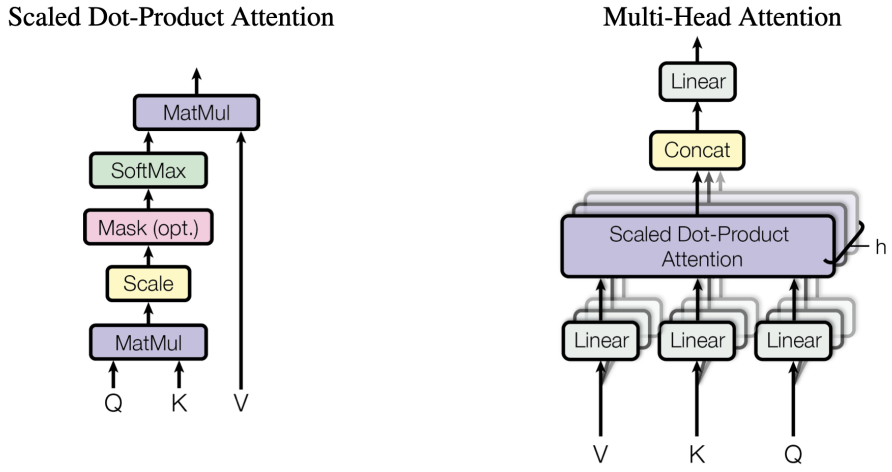

Figure 2.6: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. [69]

effectively capturing relationships between tokens regardless of their distance in the sequence. Self-Attention addresses several limitations of previous sequential models; it enables parallel computation, as the attention for each token can be calculated independently. It can capture long-range dependencies without the need for recurrent connections. The attention weights provide a degree of interpretability, showing which parts of the input are most relevant for each output. In the Transformer architecture, Self-Attention is typically used in a multi-head configuration, where multiple sets of query, key, and value transformations are learned. This allows the model to attend to different aspects of the input simultaneously. Figure 2.6 depicts the Attention mechanism.

Encoder

The encoder consists of multi-head self-attention mechanism whose output is fed into feed-forward networks. The architecture of the encoder is such that it can be stacked atop one another. The original paper [69] had 6 encoders stacked atop one another. The input to the first encoder in the stack is a sequence of vectors derived from an

embedding layer. The output of the encoder is a sequence of vectors which is passed to the decoder, described below.

Decoder

Much like the encoder, the decoder comprises multi-head self-attention and feed-forward networks. However, the self-attention mechanism in the decoder has a masking to stop it from peaking at future tokens in the sequence. This is required as the decoder also receives as input the original input vectors from the embedding layer. The output of the decoder is a probability over the vocabulary which is used to sample the token to be generated.

The Transformer architecture has found wide adoption in models like BERT [16]. BERT, which stands for Bidirectional Encoder Representations from Transformers, is an encoder-only model based on the Transformer architecture. It is pre-trained on two primary language tasks: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). These pre-training tasks are designed to help the model learn deep, contextual representations of language.

In the Masked Language Modelling task, 15% of the input tokens are masked, and the model's objective is to predict these masked tokens given their surrounding context. This approach encourages the model to develop a robust understanding of language by considering both left and right contexts when making predictions. The masking process involves replacing tokens with a special [MASK] token, random tokens, or leaving them unchanged, which helps prevent the model from relying too heavily on the mask token itself.

For the Next Sentence Prediction task, the model is presented with pairs of sentences and must determine whether the second sentence naturally follows the first in the original text. This task helps BERT learn relationships between sentences, which is crucial for many downstream tasks that require understanding of broader context or document structure.

BERT was trained on a substantial corpus of approximately 3.3 billion words, which provided it with exposure to a wide range of language patterns and structures. Initially, two main versions of BERT were released: $BERT_{BASE}$, containing 110 million parameters, and $BERT_{LARGE}$, with 340 million parameters. These models differ in their size and capacity, with the larger model generally offering higher

performance at the cost of increased computational requirements. Later, a smaller model called $BERT_{TINY}$ was introduced, containing only 4 million parameters. This compact model aims to provide a more resource-efficient option for scenarios where computational power or memory is limited, while still leveraging the benefits of the BERT architecture.

The pre-training process on such large-scale data endows BERT with the ability to capture general language understanding. This broad linguistic knowledge can then be applied to various downstream tasks through fine-tuning or feature extraction. The versatility of BERT's pre-trained representations makes it applicable to a wide range of natural language processing tasks, including but not limited to text classification, named entity recognition, question answering, and sentiment analysis.

By providing a strong foundation of language understanding, BERT has significantly impacted the field of natural language processing, enabling researchers and practitioners to achieve state-of-the-art results on many tasks with relatively minimal task-specific training. This approach of using pre-trained models as a starting point for various language tasks has become a standard practice in the field, demonstrating the power and efficiency of transfer learning in natural language processing.

Generative Pre-trained Transformer 3 (GPT-3)[7], developed by OpenAI is a significant advancement in the field of natural language processing and artificial intelligence. GPT-3 is an autoregressive language model built upon the decoder component of the Transformer architecture, which has proven to be highly effective in various language-related tasks.

Unlike BERT, it is trained to predict the next token in a sequence based on the previous tokens it has encountered, and as such can only attend to tokens to the left of the current token being processed. This left-to-right processing approach makes GPT-3 particularly well-suited for tasks that involve text generation. Such tasks include but are not limited to question answering, where the model can produce human-like responses to queries; text summarization, where it can condense longer pieces of text while retaining key information; and language translation, where it can convert text from one language to another with impressive accuracy.

The scale of GPT-3 is one of its most notable features. With 175 billion parameters, it represents a massive leap in model size compared to its predecessors and many of its contemporaries - $BERT_{LARGE}$ had 340 million parameters. GPT-3 was trained on a massive corpus of text amounting to 300 billion tokens, which encompasses a wide array of internet text, books, and other written materials.

2.2.4 Sentence Transformer

Sentence Transformers (ST) have emerged as an approach for embedding sentences into semantically rich high-dimensional vectors using Transformer models. The work, SBERT, by [55] addressed some shortcomings in the traditional BERT model when used for sentence similarity task. They modified BERT to be fine-tuned on sentence-pair tasks using a Siamese network [6]. Also, while BERT produces individual word embeddings of fixed length for each word in an input sequence, SBERT creates fixed-length vector representations at the sentence level. To accomplish this, SBERT employs pooling methods like max-pooling or mean-pooling on the transformer's final layer output, resulting in an embedding that encapsulates the entire sentence. This new model then produces embeddings which are easily compared using common similarity metrics such as cosine distance.

2.3 Literature Review of Item cold-start

In this section, we review the literature on item cold-start. First, we examine research into Matrix Factorisation models that treats item cold-start as the joint decomposition of user preferences alongside item metadata derived from TF-IDF or binary features. We also highlight some of the challenges with this approach. Then we proceed to other classes of techniques that remediate some of the challenges with a joint decomposition. And then we look at a number of advancements in Recommender Systems that uses the expressive nature of Deep Neural Networks to build robust models for item cold-start.

Relational learning via Collective Matrix Factorisation, the work by Singh et al. [63] introduced an approach to Matrix Factorisation they termed Collective Matrix Factorisation. The idea is the joint factorisation of different modalities to solve movie rating prediction. The authors modeled rating prediction as relational learning where the entities are movies and users. And these entities have metadata associated with them such as movie plots and user demography. They then extended the traditional low-rank factorisation of the typical user-movie preferences matrix to include the simultaneous factorisation of the user and item metadata matrices. By sharing the same user and item factors across the different matrices, they ensure the decomposed matrices share the same latent space.

LCE [60] jointly factorises the user rating matrix and item content matrix into

the same latent space by sharing a low-rank matrix in the factorisation. They also introduce locality by including a term which measures the similarity of adjacent points in the shared matrix. In this example, the shared matrix W is the low-rank approximation of the documents or items. In addition, they use a multiplicative update rule for optimisation.

For song recommendation, the work by Gouvert et al. [27] extended Poisson Matrix Factorisation based on listening counts to include an extra modality. Their model was able to learn from song tag labels as well. Unlike either hard or soft co-factorisation which directly penalises the coefficient matrices to varying degrees, their model included an equality constraint on the normalised coefficient matrices which takes into account the popularity of a song and the abundance of tag labels. They developed a Majorisation-Minimisation algorithm to optimise their proposed model.

The authors of [22] proposed jointly factorising a user-preference, user attribute and item attribute matrices. Since the user-preferences are implicit feedback where only the items a user has interacted with is non-zero, with most of the items being zero, they came up with a weighting scheme that depends on the similarity of the user and item attributes. Their model was optimised using Alternating Least Squares.

A 2-stage approach was developed by the authors of [53] such that the latent factors for the items were first estimated using soft-clustering via Non-negative Matrix Factorisation. The estimated latent item features are then used in another matrix factorisation to estimate the latent user features as well as predict the rank of items.

Models optimised via Ridge Regression offer some benefits such as; simplicity, a closed-form solution and computational efficiency. To that end, a number of approaches have been proposed to tackle top-N recommendations using linear models.

One such pioneering model is CSLIM [48] where they extended their work in SLIM [47] which considers top-N recommendation as a sparse aggregation of items purchased by users. That is, the decomposition of the user-item preference matrix is the product of itself and a coefficient matrix. To avoid the trivial solution where this coefficient matrix is an identity matrix, they constrain the coefficient matrix to have a diagonal of 0. Their proposed model includes the decomposition of the side information matrix and ensures it is recoverable by the same coefficient matrix used for the user-preference matrix - with the assumption that the purchasing patterns of a user on a set of items is correlated with the similarity of the properties of the items in the side information.

Olivier et al. in [34] extended the work in [65] to include side information. Their model added a term which decomposes the item-tag matrix into a product of itself and a similarity matrix, much like the original [65]. Then both are collectively solved using ridge regression. To retain the closed-form solution, they stack together the user-item preference and item-tag matrices.

The high-dimensionality of side information has been a problem for variational autoencoders [10] as it determines the size of the input, and hence dominating the size of the model. Yifan et al. [9] developed a collective variational autoencoder which collectively encodes and recovers the user-item preference and side information matrices using the same inference and generation networks. This approach addresses another shortcoming of variational autoencoders for CF, which is, having separate encode-decode networks for each modality. They proposed a 2-stage algorithm to train their model; pretrain the model using side information, then refine it with the user-item preferences.

Similar to SLIM [47], the authors of [78] learn a linear model based on the user-item preference matrix. This they combined in a joint prediction model with a linear model that learns user preferences from side information. The discriminative prediction from the side information was required to capture how different features in the item side information are relevant to different users. They developed a projected gradient ascent algorithm to solve the joint model described above.

The authors, Gartner et al. [26] developed an attribute-aware Matrix Factorisation model such that item side information is mapped to latent factors and can thus be used to overcome item cold-start. They developed two approaches; one is a mapping from item side information to latent factors using k-NN to aggregate the most similar item latent factors for an item based on item attributes similarity - this approach requires the Matrix Factorisation model to be built first. The second approach modified the BPR [57] algorithm to include a linear function of the item side information, approximating the item latent factors. Another attribute-to-feature mapping approach was the work by [13] where they consider a Gaussian distribution over possible weight matrices, as opposed to just one. In addition, they also presented an online algorithm, DynamicBPR, which is able to update the item latent factors for new items as interaction data arrives, without the need to retrain the CF model.

Items that share a common set of attributes should ideally have similar latent factors. That's the assumption by [45] in one of the approaches they proposed in their

work. This approach they named alignment-based penalty. It's a penalty because they regularise traditional matrix factorisation with a term which shrinks the latent factors of items with similar attributes close to one another. The other approach they proposed is called regression-constrained factorisation. This treats the item latent factors as a combination of a linear weight matrix and item side information.

Given the propagation of errors they noted in Collective Matrix Factorisation approaches, the authors of [4] decide to decouple the factorisation step from the knowledge transfer i.e the step where latent factors for cold start users and items are estimated using latent factors of warm users and items. They did this by carrying out matrix factorisation on a sub-matrix of the user-item preferences. This sub-matrix contains enough information about user preferences to be factorised accurately. Then they combined the estimated user and item latent factors with similarity matrices of user and item side information to generate the preferences for all users and items.

Content-based filtering approaches do not suffer from cold-start since user and item metadata are readily available. In the case of item cold-start in this instance, features can be computed from the item metadata and distance functions such as cosine similarity or Pearson correlation can be used to determine how similar items are. The same strategy can be applied for user cold-start. While useful, it does have one major challenge; the features are often high dimensional, noisy and sparse and can lead to poor recommendation. To that end, classical techniques in Information Retrieval such TF-IDF and BM25 are often used as feature selection techniques to make sure only relevant features are utilised. Broadly, TF-IDF and BM25 determine the relevance of a feature by the frequency of its occurrence in the corpus.

It should be noted that the Content-based filtering method described above does not exploit the collaborative information from user-item preferences. A new class of methods called feature weighting effectively combines these two approaches.

Elbadrawy et al. [20] developed a feature weighting suitable to sparse datasets by learning global similarity functions that captures item patterns, and combines it with user-specific weights to align each item to the user's taste better. This is a remediation of non-collaborative approaches which do not take into account global user patterns. Sharma et al. [62] extended this work by arguing that pairwise interactions of item features are better compared to independent features. To this end, they developed a factorised bilinear similarity function to model these feature interactions. A drawback of both methods as noted by Dacrema et al. [15] is that they model the filtering and exploitation of collaborative preferences jointly. They

argue that this approach is difficult to train and error propagation from either makes the final model worse. They presented a 2-stage algorithm; first an item similarity matrix is generated using a CF model in the first stage. The second stage then involves learning weights that captures the relationship between the item similarity matrix and the item features.

Algorithm	Method	Description
[60, 27, 22]	Collaborative Matrix Factorisation	Joint decomposition of user-preference matrix alongside side information matrix.
[48, 34, 78]	Sparse Linear Method	User-preference matrix and item side information can be recovered using the same coefficient matrix.
[26, 10]	Attribute-to-Feature Mapping	Maps side information to latent factors.
[45]	Alignment-based penalty	Includes a penalty term which aligns the latent factors of items with similar side information.
[4, 53]	Two-stage Matrix Factorization	Decouples recommendation into separate steps, using separate models.
[62, 15, 20]	Feature-Weighting	Uses collaborative information to learn pairwise weights for side information.

Table 2.1: Summary of Matrix Factorisation Techniques

The approaches we have looked at thus far, see Table 2.1, treat recommendations primarily as a low rank decomposition of the user-item preferences which is then augmented with a linear combination of some learned weights and features from side information to tackle item cold-start. That is to say both of these sub-processes are linear methods. Side information is also often modelled using either binary features or TF-IDF. However, there is some value to be gained from modelling user interactions and side information using non-linear methods:

- **Increased expressivity** to learn complex patterns between users, items, and side information

- Ability to include **different modalities** beyond user-item preferences
- Can handle item side information as input features for **end-to-end training** without costly pre-processing

Also, some architectures of Deep Learning lends itself easily to model inductive biases for some tasks. Consider sequence tasks such as next-item recommendations and temporal behaviour of users which are well suited to Seq2Seq models. [77] At the core of Deep Learning approaches for Recommender Systems is the Multilayer Perceptron (MLP). The perceptron is a linear combination of learned weights and some input whose output is then passed through a non-linear activation function such as sigmoid. Below, we briefly examine some of the advancements in Deep Learning for Recommender systems.

Neural Collaborative Filtering (NCF) by He et al. [30] sought to replace the inner product in classical Matrix Factorisation with an MLP.

Wide & Deep Learning for Recommender Systems by Chen et al. [10] combined memorization with generalisation. Their model is a jointly trained Generalized Linear Model and a Deep Learning model which combines the ability of the linear model to capture frequently co-occurring signals with the expressivity of the Deep Learning model, and hence superior generalisation capability. It should be noted that Deep Learning methods still fall short when the only input is the user-item preference matrix. Well tuned classical techniques like Matrix Factorisation perform much better in that scenario.

DeepFM by Guo et al. [28] combines Factorisation Machines (FM) [56] with Deep Learning. One of the disadvantages of FM is their inability to scale to large datasets with increasing higher degree of feature combinations. As such, second-order feature combinations are often used. DeepFM resolves this by combining classical FM with second-order interactions with a deep network using MLP to capture higher-order interactions. Both are trained jointly.

This chapter reviewed the literature on Collective Matrix Factorization (CMF) for item cold-start recommendations. While CMF allows the integration of metadata as side information, existing approaches use only sparse features such as binary features, count data or TF-IDF, rather than dense semantic representations from Sentence Transformers or language models. Another limitation is that auxiliary information is typically represented without considering pairwise relationships between

items, which restricts the model's ability to capture structural dependencies. These limitations reduce the effectiveness of CMF in learning meaningful item associations. The next chapter presents a model designed to address these issues.

Chapter 3

Proposed Approach

In this chapter, we describe in detail the proposed approach that solves some of the challenges with Collective Matrix Factorisation (CMF). In section 3.4, we introduce a semantics-aware approach to CMF comprised of two key elements; Symmetric Matrix Factorisation, and embeddings from Sentence Transformers. Symmetric Matrix Factorisation better captures pairwise relationships between the items, while the embeddings from the Sentence Transformer ensures the semantic nuances are captured.

3.1 Notation

We are given two sets, U, and I. U is the set of users, I, the set of items. Users will express preference for zero or more items in I. The specifics of the preference will depend on the domain. For example, it can be a rating, a like, share, save, purchase etc. This preference information is usually represented in a matrix R whose size is $|U| \times |I|$. For items where the user has not expressed any preference for the item, it is customary to set the preference to 0. In addition, we define the set I_c to be a subset of I such that no user has provided any preference information for that item.

In addition to U, and I defined above. We have a set, F. These contain the various properties or attributes that define the items. These attributes are used to create a side information matrix which we denote A, a $|I| \times |F|$ matrix where each row represents an item and the columns - the relevant attributes of that item.

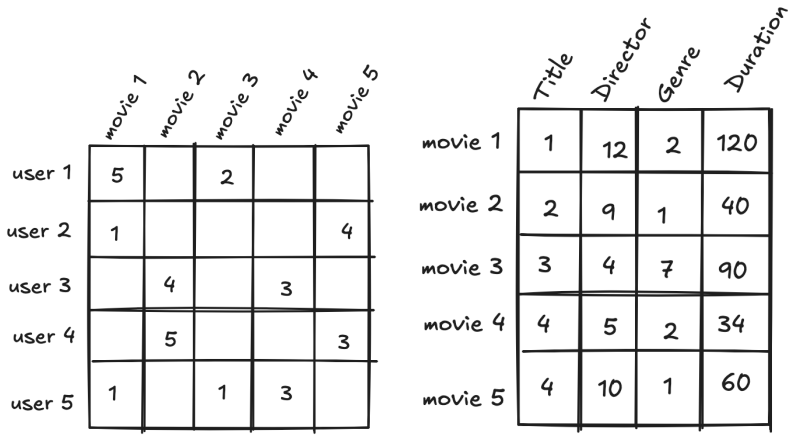

Figure 3.1: Left: user rating matrix. Right: item content matrix

3.2 Collective Matrix Factorisation

Singh and Gordon [63], define user preferences for items by:

$$\min_{P,Q} ||R - PQ^T||_F^2 + ||A - QS^T|| \tag{3.1}$$

P, Q and S are latent factors that describes the user preference for a particular item given the observed preference and any item related metadata captured by the matrix, I. R can for example be a binary matrix of observed user preference on movies or the ratings on a scale of 1-5 given by users to movies. This preference matrix is often quite sparse as users only ever see or interact with a handful of the entire catalogue of movies. And I can be a feature matrix of item metadata such as genre, plot, cast etc. These feature matrix is often sparse as it is represented using one-hot encoding or TF-IDF. The item latent factors from the factorisation of R can capture item characteristics such as genre, mood or language. The latent factors for a user then captures to which extent the user agrees with these characteristics of the item, indicating their preference or otherwise. While each matrix can be factorised independently, we will not be taking advantage of the relationship between both data modalities. The item characteristics discovered by the factorisation of the user preference matrix are often available as metadata that can be incorporated directly. Thus, the crux of Collective Matrix Factorisation is exploiting such relationships by representing both matrices in a common latent space.

This approach of jointly decomposing the user preference and side information matrices was explored in the work by Saveski and Mantrach [60]. The optimisation problem they sought to solve was given as:

$$J = \frac{1}{2}[\alpha||X_s - WH_s||^2 + (1-\alpha)||X_u - WH_u||^2 + (W^T LW)] \tag{3.2}$$

The regularisation terms are omitted for brevity. In the formulation above, X_s is the side information matrix, which for them was a document-term matrix, while X_u captures if a user has interacted with a document. In addition, to enforce the local geometric structure of the original data in the low-dimensional space, they build an adjacent matrix of p nearest neighbours for each data point in the original high-dimensional space. This adjacency matrix is then used to weight the low-dimensional representation of the data points. The weighting scheme ensures that points which are close to each other in the original high-dimensional space maintain their proximity in the new low-dimensional representation. The computation of the adjacent matrix introduces a hyper-parameter, p and happens outside of the main factorisation, thus we regard this as a 2-stage approach. Their approach also has the constraint that the weights W, H_u and H_s are non-negative. This constraint limits the range of input their algorithm can accept, and it is on this our proposed approach builds on, amongst other things.

Our proposed method revisits how the side information is incorporated into Collective Matrix Factorisation, while leaving the preference matrix largely intact. We describe our proposed method below.

3.3 Side Information Decomposition

Given the side information matrix, A, we wish to find clusters such that items with similar metadata are grouped together. Approaches to clustering include k-means [25], DBSCAN [21] and Spectral Clustering [70]. However, for our purposes, we consider a factorisation-based approach called Non-negative Matrix Factorisation (NMF) [49].

$$A = WH \tag{3.3}$$

The side information matrix, A, can be decomposed into two matrices W and H. W is an $|F| \times k$ matrix, known as the basis or feature matrix. While H is the

coefficient matrix, with dimensions $k \times |I|$. k denotes the rank of the matrix, and is also the number of clusters identified. The matrices W and H are non-negative, as is the input.

A is a document-term matrix which for the purpose of this work is either a Term Frequency-Inverse Document Frequency matrix (TF-IDF) or sentence embeddings matrix.

Kim et al. [36], when sparsity is imposed on H, show how this is equivalent to k-means. In this formulation, W is equivalent to the centroids, and H indicates which cluster each item belongs to by selecting the index with the largest value. They introduced a parameter, β, which determines the degree of sparsity of H - with larger values indicating stronger sparsity and being closer to hard clustering. In addition, the size of the elements of W are controlled using a parameter, η. Taken together, these two terms can be seen as regularisers. This formulation of NMF imposes a non-negativity constraint on the matrix A, as well as its factors, W and H. However, Ding et al. [17], introduce various forms of NMF where the non-negativity constraint is relaxed. One such form is what they call semi-NMF which is described thus:

$$A_{\pm} \approx W_{\pm} H_{+}^{T} \tag{3.4}$$

Semi-NMF has the property that the input and the basis/features matrices are unconstrained ($\pm$) i.e. the values in the matrices can be a mix of positive and negative numbers, while the non-negativity constraint is retained for the coefficient matrix. They find that this formulation still retains the clustering ability. The relaxation of this non-negativity is crucial for our work because our side information matrix is composed of embeddings from a Sentence Transformer and as such have mixed signs. Sentence Transformers generates contextualized embeddings that capture semantic relationships in a high-dimensional space. These embeddings naturally contain both positive and negative values, as they represent complex linguistic patterns through distributed representations. Non-negative Matrix Factorisation assumes non-negativity, which works well for count-based data or strictly positive features. However, this assumption becomes restrictive when dealing with pre-trained language models like Sentence Transformers.

Symmetric Matrix Factorisation (SMF) [39] [76] presents a novel approach by reinterpreting Non-negative Matrix Factorisation (NMF) as the decomposition of a pairwise similarity matrix. Unlike traditional NMF, which focuses on factorising the

data matrix into a set of basis vectors and corresponding coefficients, SMF emphasises the relationships between data points by decomposing the similarity matrix. This similarity matrix is essential as it captures the underlying relationships and distances between various data points within the dataset.

As noted by Kuang et al. [39], while NMF has gained popularity and has been effectively applied to clustering tasks [72], often yielding superior results compared to classical clustering methods like k-means [36], it is important to recognise that NMF is not a one-size-fits-all solution for every clustering problem. The primary assumption behind NMF is that each cluster within the data can be accurately represented by a basis vector. Consequently, the original data matrix can be approximated through a linear combination of these basis vectors. This linearity assumption, however, imposes a limitation on NMF, as it fails to capture the structure of clusters when they exhibit non-linear relationships. In scenarios where the data's inherent clusters do not conform to a linear structure, the effectiveness of NMF diminishes, highlighting the need for alternative approaches like SMF. This formulation of SMF is then given as:

$$A_{\pm} \approx HH^T \tag{3.5}$$

3.4 Semantics-aware Collective Matrix Factorisation

We present Semantics-aware Collective Matrix Factorisation (SaCMF), an approach based on Collective Matrix Factorisation that uses Symmetric Matrix Factorisation to cluster similar items. We transform the side information matrix, A, into a pairwise similarity matrix by taking its dot product with its transpose, resulting in a symmetric matrix.

$$A^* = AA^T \tag{3.6}$$

This transformation encodes the pairwise relationships across items. The crux of our method follows from the cluster hypothesis which says "closely associated documents tend to be relevant to the same requests" [58]. So we envision a scenario where deriving the latent item features from the decomposition of the side information alongside the user preferences will result in better performance on item

cold-start. This is because the pairwise relationship encoded in the side information matrix allows cold items to benefit from the embeddings learned for warm items as the latter has sufficient collaborative information. Unlike regular Matrix Factorisation, Symmetric Matrix Factorisation, depicted in Figure 3.2, is the product of a low-rank matrix and its transpose.

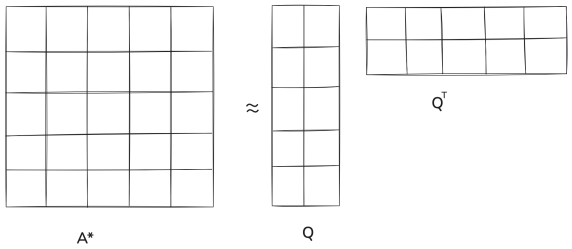

Figure 3.2: Symmetric Matrix Factorisation

Following the Collective Matrix Factorisation approach, we combine the methods described above to arrive at our proposed approach. Because the focus of our work is implicit feedback, and as such the preference matrix is a binary matrix, we use Binary Cross Entropy as the loss function in the decomposition of the user-item preference matrix. For the decomposition of the item content matrix, we retain Frobenius norm as the loss function.

$$\hat{r}_{ui} = \sigma(p_u q_i^T) \tag{3.7}$$

$$\min_{P,Q} \left[-\sum_{u,i} \left(r_{ui} \log(\hat{r}_{ui}) + (1 - r_{ui}) \log(1 - \hat{r}_{ui}) \right) + ||A^* - QQ^T||_F^2 + \lambda \left(||P||_2^2 + ||Q||_2^2 \right) \right] \tag{3.8}$$

r_{ui} is the actual rating a user, u, given to an item, i, while $\hat{r}_{ui}$ is the predicted rating. We also include regularisation terms in our objective function described above; L_2 norm for the user and item factors.

One interpretation of the second term above is as an alignment penalty. This penalty term minimizes the discrepancy between the item content pairwise similarity matrix and learned item factors pairwise similarity matrix. This encourages the factorisation of the rating matrix to learn representations that preserve the underlying content relationships between items, while maintaining flexibility in how these

relationships are encoded.

In this chapter, we presented a model that enhances Collective Matrix Factorisation through two modifications. First, we incorporate embeddings from Sentence Transformers as features for the side information. This allows the model to utilise rich semantic relationships in the provided item metadata. Secondly, side information is represented as a symmetric matrix which allows us to capture pairwise relationships among items. This approach improves traditional CMF by integrating rich features derived from Sentence Transformers while maintaining a structure that preserves important relational information between items.

In subsequent chapters, we evaluate our model against a number of real-world datasets and compare it against various baseline models. The evaluation allows us to empirically show the effectiveness of our proposed model. We start by introducing the datasets and the experiment protocol, then we discuss the results of the experiment.

Chapter 4

Experimental Setup

In this section we describe the source of data, the data and its attribute as well as the methodology we will be using. We also highlight the metrics we will use for evaluation and the baselines we will be comparing our proposed method against.

4.1 Datasets

We used the popular Movielens 1M [29] and Yahoo! Movies [73] datasets for evaluation. Movielens 1M dataset was collected from the year 2000 to 2003 from the Movielens website. This was a project ran by the GroupLens research team based out of University of Minnesota. The Movielens 1M dataset contains $1,000,029$ ratings given by $6,040$ users to $3,076$ movies. These ratings are on a 1-5 scale, but for our purposes we discretise the ratings so that ratings greater than 3.5 are set to 1 to signal a user preference for that movie, and 0 otherwise. These movies cover 12 genres. For side information, we fetch the associated plot for each movie from IMDB using the provided movie ID in the files downloaded from the Movielens website. Yahoo! Movies dataset is a sample of the movie preferences of members of the Yahoo! Movies community. This dataset provides the movie plot for the $3,679$ movies in its catalogue. Unlike, Movielens, the Yahoo! Movies dataset only has $197,000$ ratings by $7,619$ users.

In the datasets, movies typically belong to multiple genres, which is to be expected as different themes can be observed in the same movie. To plot the chart in Fig. 4.1, we select the first genre in the list of genres for a movie. As the chart shows,

Dataset	Users	Items	Ratings (k)	Density (%)
Movielens 1M	6040	3706	1000	4.5
Yahoo! Movies	7619	3679	197	0.7

Table 4.1: Details of datasets used

Comedy and Drama dominates the genres, with Action and Horror next.

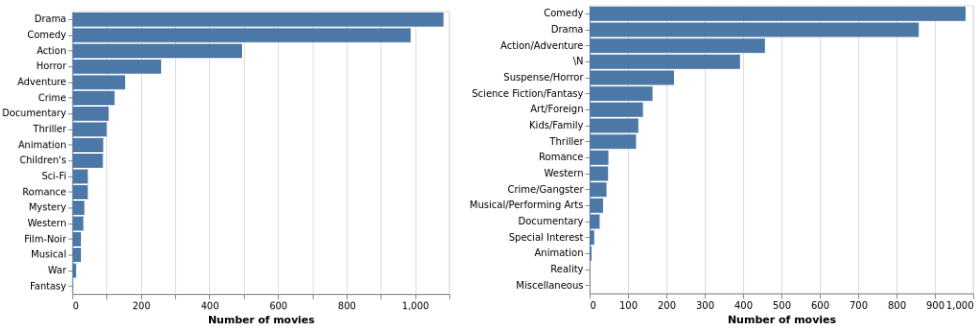

Figure 4.1: Left: Genres in Movielens 1M. Right: Genres in Yahoo! Movies

4.2 Data Preprocessing

We employ two approaches for encoding the movie plot: Term Frequency - Inverse Document Frequency (TF-IDF), and embeddings from sentence transformers. For sentence embeddings, we use Transformer-based language model, Qwen2-1.5B-Instruct, described below, that map sentences and paragraphs to a high-dimensional space using dense vectors. These vectors encode the meaning of the text and can be used for various applications such as clustering similar texts or performing semantic search tasks.

4.2.1 Qwen2-1.5B-Instruct

This is a decoder-only large language model developed by Alibaba which belongs to the Qwen2 [74] group of models ranging in size from 0.5 billion to 70 billion parameters. We settled on the 1.5B parameter model because of its strong performance on the clustering task in the Massive Text Embedding Benchmark[44]. This model was

Star Wars: Episode II Ten years after the events of THE PHANTOM MENACE, not only has the galaxy undergone significant change, but so have our familiar heroes Obi-Wan Kenobi (Ewan McGregor), Padmé Amidala (Natalie Portman) and Anakin Skywalker (Hayden Christensen) as they are thrown together again for the first time since the Trade Federation invasion of Naboo. Anakin has grown into the accomplished Jedi apprentice of Obi-Wan, who himself has transitioned from student to teacher. The two Jedi are assigned to protect Padmé whose life is threatened by a faction of political separatists. As relationships form and powerful forces collide, these heroes face choices that will impact not only their own fates, but the destiny of the Republic.
Titanic An undersea expedition searching for a valuable diamond aboard the wreckage of the Titanic instead finds a drawing of seventeen-year-old Rose DeWitt Bukater, on the way to her wedding to a wealthy tycoon. While Rose falls in love with Jack Dawson, a free-spirited artist and third-class passenger who ignites the unquenchable fires of passion inside her, the hubris of the shipś crew tempts them to test the cross-Atlantic speed record–smack into an iceberg. A rightfully celebrated, no-holds-barred, boffo blockbuster, with enough heart and soul to balance its extravagant special effects and record-breaking budget.Academy Award Nominations: 14, including Best Actress (Winslet) and Best Supporting Actress (Stuart).Academy Awards: 11, including Best Picture, Best Cinematography, Best Original Dramatic Score, Best Visual Effects, Best Costume Design, and Best Song ("My Heart Will Go On," by James Horner and Will Jennings - performed by Celine Dion)

Table 4.2: Movie Plots for Star Wars: Episode II and Titanic

pre-trained on 7 trillion tokens to ground it in natural language understanding and text generation tasks. After pre-training, this model was then instruction-tuned on a range of tasks such as coding, mathematics, logical reasoning, instruction following, and multilingual comprehension. The output of the model was also aligned with human values by post-training using Reinforcement Learning from Human Feedback [12] on preference data that had been collected.

This model has a context length of 32k tokens, allowing it to process large input, and it uses Grouped Query Attention that makes it faster and use less memory during inference. The movie plots are processed by this model to produce a 1536-dimensional sentence embedding for each plot.

4.2.2 TF-IDF

To create the TF-IDF features, we use the TF-IDF Vectoriser in the scikit-learn pacakge. We remove stopwords using the default list of words provided by the package, and ignore words that appear in fewer than three documents. The dimension of the resulting plot vector is not fixed like the Transformer models above, but depends on the vocabulary of the corpus. The sentence vector for Movielens dataset contains $9,933$ entries, while the vector for Yahoo! Movies dataset has $10,657$ entries.

4.3 Baselines

We compare our proposed model with the state of the art in Collective Matrix Factorisation.

Local Collective Embedding (LCE) [60] does collective factorisation on the preference and side information matrices. In addition, it constructs a graph of nearest neighbours which it uses to enforce locality via regularisation in the previous factorisations. It does this so that latent factors of items having similar attributes are close in this low-dimensional space resulting from the factorisation. In that case, we regard it as a 2-stage approach.

ItemKNN-CBF [46] is a memory based approach, and thus has no weights. Instead, recommendations are based on similarity between items via their content features using k-Nearest Neighbours.

4.4 Data Partitioning

An important consideration during the split is ensuring that interactions in the validation and test sets involve users present in the training set. This eliminates the confounding factor of user cold start, where a model might struggle to recommend for entirely new users unseen in the training data. Our focus here is solely on the model's ability to recommend for new items, even if the users interacting with them are familiar to the system.

Our evaluation methodology closely follows the approach established by Dacrema et al. [15]. We employ a common train-validation-test split strategy to assess the model's performance, particularly its ability to handle item cold start issues. This split involves randomly selecting 60% of the items and their interactions for the training set. The remaining 40% is then divided equally (20% each) into validation and test sets. This ensures a clear separation between the sets, mimicking a real-world scenario where new items are constantly being introduced (item cold start).

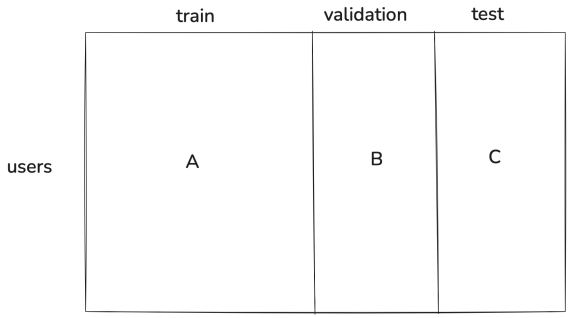

Figure 4.2: User rating matrix split. A has warm items, while B and C contains cold items.

4.5 Evaluation

To gauge the quality of the recommendations returned by our model, we require a scoring function that compares the predicted recommendations against a relevance judgement list. We consider two approaches for evaluation: classification metrics, and ranking metrics.

When evaluating a recommender system, we often have a limit k, which denotes how large the list of predicted recommendations should be. This is particularly useful because recommendations are often presented on user interfaces with limited real estate, so we only need to evaluate the performance for a short list. We use k below to signify such a limit.

4.5.1 Classification Metrics

Classification metrics compare the predicted recommendations against a binary judgement. Binary judgements are implicit feedback for example; click, purchase, and visit. In this case, the goal of classification metrics is to determine the proportion of the predicted recommendations that are relevant to the user. Below, we look at two such metrics.

Precision

Precision is the proportion of the predicted recommendations that are actually relevant. It is defined thus:

$$P@k = \frac{\#\text{ relevant recommendations}}{k} \tag{4.1}$$

The possible values for precision are between 0 and 1, in order of increasing quality.

Precision does not account for the order of the predicted recommendations, and this is a shortcoming. Imagine a list of predicted recommendations such that the most relevant items appear at the bottom of the list, and thus are presented to the user last. This behaviour is remedied by ranking metrics which we shall take a look at below.

Recall

Recall, measures the proportion of all relevant items returned in the predicted recommendations. Possible values are between 0 and 1. It is defined thus:

$$R@k = \frac{\#\text{ relevant recommendations}}{\text{relevant items}} \tag{4.2}$$

This can be interpreted as the probability that a relevant item is predicted by our model. Recall also has the same shortcoming with Precision in that it does not take the order of the predicted recommendations.

4.5.2 Ranking Metrics

Unlike classification metrics where the order of the predicted recommendations does not matter, ranking metrics prioritises scenarios where the most relevant items are ranked higher in the list, reflecting a user's likelihood of interacting with them. We consider two such metrics below.

Normalised Discounted Cumulative Gain (NDCG)

At the heart of NDCG is Cumulative Gain, which counts the number of relevant items in the top k results. Possible values are between 0 and 1. This is formulated as follows:

$$CG@k = \sum_{i=1}^{k} rel_i \tag{4.3}$$

rel_i above is 1 if the item at position i is relevant, otherwise 0. NDCG also has the useful property in that relevance can be on scale e.g from 1 to 5, in order of increasing relevance. Ideally, what we want is for relevant items to appear at the top of the list. For this to happen, we can weight each item in the returned list in decreasing order as its position increases. Hence, a relevant item at position 1 in the returned list will be weighted higher than a relevant item at position 10 in the list. This is referred to as Discounted Cumulative Gain and is formulated as below:

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{4.4}$$

To normalise this score, we compute Ideal Discounted Cumulative Gain (IDCG), which is the maximum possible score produced by sorting the items according to their ground truth relevance. IDCG is given as:

$$IDCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{4.5}$$

Then we compute NDCG@k as follows:

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{4.6}$$

Mean Average Precision (MAP)

Similar to NDCG, Mean Average Precision computes a score which indicates the quality of ranked recommendations. However, unlike NDCG, it has the limitation that it requires the relevance to be binary. Possible values for MAP are between 0 and 1. MAP extends precision to account for rank by taking the average precision of values at all relevant positions. Taking the mean of the average precision for all users then gives us MAP. Average Precision (AP) is defined thus:

$$AP@k = \frac{\sum_{i=1}^{k} Precision_i.rel_i}{\text{total number of relevant items}} \tag{4.7}$$

rel_i is the relevance at position i, $Precision_i$ is the precision at position i of the ranked list. MAP is thus:

$$MAP@k = \frac{1}{N}\sum_{k=1}^{n} AP@k \tag{4.8}$$

4.6 Implementation

4.6.1 Technologies

The proposed model was written in PyTorch[1] and extensive experiments carried out on an Ubuntu 20.04 machine with the following specifications:

- Core i9 CPU
- 32GB RAM
- Nvidia GPU RTX 3090 Ti with 32GB RAM

In addition, a suite of ancillary libraries were used.

[1]The source code is publicly available at https://github.com/farouqzaib/pycmf

- SBERT [55]: this is a Python library that provides access to state-of-the-art models for computing sentence embeddings.
- scikit-learn [51]: a Python library providing various Machine Learning algorithms as well as data processing tools.
- Huggingface [71]: a Python library that provides access to state-of-the-art Transformer models.

4.6.2 Hyper-parameter Tuning

The training and validation sets play a crucial role in optimizing the model's performance. We utilize these sets for hyper-parameter tuning, a process where we adjust various model settings to achieve the best possible results. This is often done through cross-validation, where the training set is further split into smaller subsets for training and validation purposes. By iteratively evaluating the model with different hyper-parameter configurations on these smaller sets, we can identify the optimal settings for the final model.

Once the hyper-parameters are selected using the combined training and validation sets, the final model is trained on this larger dataset. This "tuned" model is then used to predict the relevance scores for items in the unseen test set. To account for potential variations in model performance due to random sampling during the data split, we repeat this entire experiment five times. The final reported performance metrics in this paper represent the average results obtained across these five repetitions. This practice helps ensure the robustness and generalisability of our evaluation findings.

The main hyper-parameters are the user and item factors learning rates, L2 regularisation and the number of latent factors. In section 5.2, we provide an analysis of the hyper-parameters.

Chapter 5

Results and Analysis

This section reports the results of the experiments we conducted on two datasets: Movielens 1M and Yahoo! Movies. To assess the effectiveness of our approach in handling item cold start issues, we first established a set of quantitative evaluation metrics commonly used in recommendation systems; MAP, Precision, Recall and NDCG. We then evaluated the model's performance on one key aspect:

Nearest Neighbour Quality: We analysed the quality of nearest neighbours identified for a few cold start items. Accurate nearest neighbours are essential for recommending relevant items to users who have not interacted with many movies yet.

Following the evaluation, we discuss the overall effectiveness of the model in handling item cold start problems. The results, summarised in Tables 5.1 - 5.4, showcase the effectiveness of our approach. We show the mean of 5 runs alongside the margin of error calculated at the 95% confidence interval.

Algorithm	Movielens 1M			
	Precision@10	Precision@20	Recall@10	Recall@20
LCE	0.053 ±0.012	0.045 ±0.010	0.024 ±0.007	0.036±0.009
ItemKNN-CBF	0.062 ±0.020	0.050 ±0.012	0.030 ±0.011	0.040 ±0.012
ItemKNN-CBF (ST)	0.114 ±0.029	0.095 ±0.021	0.042 ±0.011	0.070 ±0.015
SaCMF	0.067 ±0.014	0.056 ±0.010	0.026 ±0.008	0.041 ±0.011
SaCMF (ST)	**0.139** ±0.024	**0.116** ±0.017	**0.051** ±0.010	**0.082** ±0.015

Table 5.1: Precision and Recall for Movielens 1M

Algorithm	Movielens 1M			
	MAP@10	MAP@20	NDCG@10	NDCG@20
LCE	0.022 ±0.005	0.014 ±0.003	0.048 ±0.015	0.049±0.015
ItemKNN-CBF	0.029 ±0.012	0.018 ±0.007	0.059 ±0.025	0.058 ±0.022
ItemKNN-CBF (ST)	0.058 ±0.021	0.038 ±0.013	0.107 ±0.032	0.105 ±0.028
SaCMF	0.032 ±0.007	0.020 ±0.004	0.059 ±0.016	0.058 ±0.016
SaCMF (ST)	**0.076** ±0.021	**0.052** ±0.013	**0.129** ±0.028	**0.126** ±0.025

Table 5.2: MAP and NDCG for Movielens 1M

Algorithm	Yahoo! Movies			
	Precision@10	Precision@20	Recall@10	Recall@20
LCE	0.025 ±0.008	0.019 ±0.005	0.053 ±0.016	0.076 ±0.016
ItemKNN-CBF	0.031 ±0.014	0.022 ±0.009	0.063 ±0.025	0.087 ±0.032
ItemKNN-CBF (ST)	0.038 ±0.012	0.027 ±0.008	0.077 ±0.029	0.110 ±0.036
SaCMF	0.031 ±0.015	0.022 ±0.008	0.060 ±0.025	0.085 ±0.027
SaCMF (ST)	**0.039** ±0.016	**0.030** ±0.010	**0.082** ±0.028	**0.123** ±0.032

Table 5.3: Precision and Recall for Yahoo! Movies

Algorithm	Yahoo! Movies			
	MAP@10	MAP@20	NDCG@10	NDCG@20
LCE	0.011 ±0.004	0.006 ±0.002	0.045 ±0.0156	0.053 ±0.016
ItemKNN-CBF	0.015 ±0.006	0.008 ±0.003	0.057 ±0.024	0.065 ±0.026
ItemKNN-CBF (ST)	**0.018** ±0.007	**0.010** ±0.004	**0.068** ±0.023	0.078 ±0.025
SaCMF	0.014 ±0.007	0.008 ±0.004	0.054 ±0.025	0.062 ±0.026
SaCMF (ST)	0.017 ±0.008	**0.010** ±0.004	**0.068** ±0.028	**0.081** ±0.029

Table 5.4: MAP and NDCG for Yahoo! Movies

Models trained with features from a Sentence Transformer have the suffix "(ST)", otherwise the features are TF-IDF. The results in bold face are the best performing, while the ones underlined are the second best.
Due to the non-negativity constraint in LCE, we were limited to testing it only with TF-IDF features. The Sentence Transformer embeddings, which contain negative values, were not compatible with LCE without additional processing steps. In contrast, we were able to evaluate both ItemKNN-CBF and our proposed SaCMF model using both TF-IDF and Sentence Transformer (ST) features, allowing for a more comprehensive assessment of their capabilities.
The experimental results demonstrate that SaCMF consistently outperforms LCE across all evaluation metrics. This superior performance validates our approach of using Symmetric Matrix Factorisation for processing side features. The improvement suggests that our method more effectively captures and utilises the pairwise relationships between items and their associated features.
When examining performance on specific datasets, we found varying degrees of improvement. On the Yahoo! Movies dataset, SaCMF shows comparable performance to ItemKNN-CBF in terms of MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain) metrics, while achieving modest improvements in Precision and Recall scores. These results indicate that our method is at least as effective as existing approaches for this particular dataset. The most substantial improvements were observed on the Movielens 1M dataset, where SaCMF demonstrated superior performance across all evaluation metrics. Particularly noteworthy is the 18% improvement in NDCG compared to ItemKNN-CBF, which was the next best performing model.

5.1 Neighbourhood Analysis

In Table 5.5, we show two movie titles and a list of five nearest neighbours each of the algorithms returned. Embedding similarity is measured using Cosine distance. The nearest neighbours are listed in descending order of similarity. For this particular analysis, the subject movies were selected from the test set to ensure an unbiased evaluation of the algorithms' performance. As seen from the table, LCE, one of the baseline algorithms, struggled to find all the relevant titles for Star Wars. LCE managed to find only two titles belonging to the series of Star Wars movies. On the

other hand, the top-5 movies returned by our algorithm belongs to the Star Wars franchise. For Aladdin, LCE again failed again to find relevant neighbours. This is not the case for our algorithm as the top-5 titles returned are titles of animated films with similar themes as Aladdin and also produced by Walt Disney. For both titles, ItemKNN-CBF performed well, as most of the titles returned are relevant to the query title.

Algorithm	Nearest Neighbours	
	Aladdin	Star Wars
LCE	This Boy's Life Asoka Dragon: The Bruce Lee Story Wasabi Battlefield Earth	Star Wars: Episode II Star Wars II (IMAX) The Art of War Driving Miss Daisy An Officer and a Gentleman
ItemKNN-CBF (ST)	Pocahontas Cinderella The Muppet Movie Robin Hood The Great Muppet Caper	Star Wars: Episode II The Empire Strikes Back Star Wars II (IMAX) The Lord of the Rings:- The Two Towers Return of the Jedi
SaCMF (ST)	Pocahontas Robin Hood (1973) The Hunchback of Notre Dame Hercules (1997) The Rescuers	Star Wars: Episode II Star Wars II (IMAX) The Empire Strikes Back Star Wars: Episode 1 Return of the Jedi

Table 5.5: 5 Nearest neighbours of the Aladdin and Star Wars movies for each algorithm

One key advantage of our approach lies in its efficiency. Unlike LCE, which requires a separate step to compute nearest neighbours and introduce locality, our method leverages the factors derived by Symmetric Matrix Factorisation (SMF) to directly estimate user preferences. This eliminates an extra computational step and potentially improves overall model efficiency, making it suitable for large-scale recommendation tasks.

Our method effectively incorporates additional movie information by using synopsis data processed through Sentence Transformers. This provides a richer understanding of each movie's content and themes. In contrast, the LCE approach cannot directly utilise this valuable data source because of its non-negativity constraint, which would require additional processing steps for the Sentence Transformer embeddings. By leveraging embeddings from Large Language Models, our system can understand more subtle and complex aspects of movies that are not captured in simple user-item interaction data. This results in recommendations that take into account not just viewing patterns, but also the actual content and thematic elements of the movies, leading to more informed and comprehensive suggestions for users.

It is important to acknowledge that joint factorisation models, including ours and LCE, can be susceptible to error propagation as noted in prior research [53]. This means that errors in one part of the model can propagate to other parts, potentially affecting recommendation accuracy. However, despite this drawback, our model demonstrably outperforms on Movielens 1M dataset, a strong baseline like ItemKNN-CBF which has been shown to outperform Deep Learning models [24] [23], showcasing the benefits of a unified framework that leverages both user-item interactions and side information.

In conclusion, the experimental results on both Movielens and Yahoo! Movies datasets validate the effectiveness of our proposed recommendation approach. It achieves superior performance on the Movielens dataset and demonstrates competitive results on Yahoo! Movies. The model's efficiency, ability to leverage side information, and unified framework position it as a promising solution for recommendation tasks, especially when dealing with cold start items.

5.2 Parameter Analysis

The SaCMF model has two key parameters: k (number of latent variables) and λ (regularization strength). Parameter k controls model complexity. Small k causes underfitting, while large k causes overfitting. Looking at the values in Table 5.6 below, the models all favour a high value of k.
Parameter λ controls regularization. Values between 0 and 1 work best, as too high λ oversimplifies the model while too low λ allows overfitting. Appropriate regularisation helps create stable solutions. Both of the findings above mirrors those

reported in LCE [60]. Unlike LCE, we do not weight the task, but rather used the full losses as we found weighting to be ineffective in our experiments. Finally, for the item content pairwise similarity matrix, using the complete matrix performed better than k-nearest neighbours approaches. This shows that even weak similarity relationships contain useful information for the model.

The final parameters for the experiments are listed in Table 5.6 for each side information type for each of Yahoo! Movies (YM) and Movielens 1M (M1M). η_u and η_i are respectively, the user and item factors learning rates. λ is L2 regularisation. k, is the number of latent factors. The table also shows the number of epochs for each model.

Hyper-param	Range	YM (TF-IDF)	YM (ST)	M1M (TF-IDF)	M1M(ST)
η_u	[1e-4, 1e-3]	1e-3	8e-4	8e-4	1e-3
η_i	[1e-5, 8e-5]	1e-5	1e-4	1e-5	3e-5
λ	[1e-5, 1e-1]	7e-4	7.5e-4	1e-3	1e-5
k	[50, 500]	500	432	500	500
Epoch	[50]	50	50	48	45
Batch size	[64, 1024]	128	64	128	128

Table 5.6: Hyper-parameter configuration and final values

5.3 Running Time Analysis

Training and inference of the model was performed on a GPU. Figure 5.1 shows how long it took for one epoch of each model, for the various side information. Movielens contains about five times more data than Yahoo! Movies but only took about 1.7 times longer when using TF-IDF. And 1.5 times longer when using features from a Sentence Transformer.

Figure 5.1: Training times per epoch

Chapter 6

Conclusion

In this thesis, our primary objective was to explore the integration of Collective Matrix Factorisation with from Sentence Transformer for feature extraction to address the item cold start problem in two movie datasets. We utilised a Sentence Transformer to encode the movie plots, allowing us to capture the semantic structure of the text. This approach provided a richer and more complete feature set compared to traditional methods of text representation. By using a Sentence Transformer, we were able to extract nuanced information from the plot summaries, including themes, character dynamics, and narrative structures that might be relevant to user preferences.

In Chapter 3 of the thesis, we presented a novel reformulation of Collective Matrix Factorisation. Our approach involved the joint factorisation of two key components; a user preference matrix, representing the interactions between users and movies, and a decomposition of pairwise interactions of item features, specifically the movie plot embeddings derived from a Sentence Transformer. This reformulation allowed us to incorporate the rich semantic information extracted by the Sentence Transformer into the collaborative filtering framework of CMF. By doing so, we aimed to create a more robust model that could better handle the item cold start problem.

It is important to note that our approach is not limited to the specific Sentence Transformer we explored in this work. The framework we developed is designed to be flexible and can accommodate various types of Sentence Transformers. This generalisation allows for future improvements as new and more advanced language models become available.

We conducted a comprehensive evaluation of our model using the two movie

datasets mentioned earlier. Our evaluation metrics focused on the model's ability to make accurate recommendations, particularly for new items with limited user interaction data.

The results of our evaluation demonstrated that our approach outperformed the LCE (Local Collective Embedding) method, which is a strong baseline for CMF in the field of recommender systems. This performance improvement was observed across various metrics, indicating the robustness of our approach in addressing the item cold start problem.

While our current results are promising, there are several avenues for future research and improvement. We plan to extend our experiments to include other types of data for item features, such as user reviews, critic reviews, or metadata about the movie's production. As well as consider entirely new domain such as Amazon Reviews and Goodbooks dataset. Additionally, we intend to modify the factorisation of the user preference matrix to consider pairwise interactions, similar to the approach we explored for item features. This could potentially capture more complex patterns in user behaviour and preferences. Given the rapid advancements in language model technology, we aim to experiment with newer Sentence Transformers as they become available, potentially improving the quality of our feature extraction. As we move towards larger datasets, it will be crucial to investigate the scalability of our approach and develop optimizations for handling vast amounts of data efficiently.

Bibliography

[1] Abdollahpouri, H., Burke, R., and Mobasher, B. Controlling popularity bias in learning-to-rank recommendation. In *Proceedings of the eleventh ACM conference on recommender systems* (2017), pp. 42–46.

[2] Ahn, J.-w., Brusilovsky, P., Grady, J., He, D., and Syn, S. Y. Open user profiles for adaptive news systems: help or harm? In *Proceedings of the 16th international conference on World Wide Web* (2007), pp. 11–20.

[3] Balabanović, M., and Shoham, Y. Fab: content-based, collaborative recommendation. *Communications of the ACM 40*, 3 (1997), 66–72.

[4] Barjasteh, I., Forsati, R., Masrour, F., Esfahanian, A.-H., and Radha, H. Cold-start item and user recommendation with decoupled completion and transduction. In *Proceedings of the 9th ACM Conference on Recommender Systems* (2015), pp. 91–98.

[5] Bell, R., Koren, Y., and Volinsky, C. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (2007), pp. 95–104.

[6] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems 6* (1993).

[7] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R.,

Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020.

[8] Canny, J. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval* (2002), pp. 238–245.

[9] Chen, Y., and de Rijke, M. A collective variational autoencoder for top-n recommendation with side information. In *Proceedings of the 3rd workshop on deep learning for recommender systems* (2018), pp. 3–9.

[10] Chen, Y., Zhao, X., and de Rijke, M. Top-n recommendation with high-dimensional side information via locality preserving projection. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2017), pp. 985–988.

[11] Choudhary, S., Koul, S., Mishra, S., Thakur, A., and Jain, R. Collaborative job prediction based on naïve bayes classifier using python platform. In *2016 international conference on computation system and information technology for sustainable solutions (CSITSS)* (2016), IEEE, pp. 302–306.

[12] Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. *Advances in neural information processing systems 30* (2017).

[13] Cohen, D., Aharon, M., Koren, Y., Somekh, O., and Nissim, R. Expediting exploration by attribute-to-feature mapping for cold-start recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems* (2017), pp. 184–192.

[14] Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems* (2016), pp. 191–198.

[15] Dacrema, M. F., Gasparin, A., and Cremonesi, P. Deriving item features relevance from collaborative domain knowledge. *arXiv preprint arXiv:1811.01905* (2018).

[16] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[17] Ding, C. H., Li, T., and Jordan, M. I. Convex and semi-nonnegative matrix factorizations. *IEEE transactions on pattern analysis and machine intelligence 32*, 1 (2008), 45–55.

[18] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research 12*, 7 (2011).

[19] Eksombatchai, C., Jindal, P., Liu, J. Z., Liu, Y., Sharma, R., Sugnet, C., Ulrich, M., and Leskovec, J. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 world wide web conference* (2018), pp. 1775–1784.

[20] Elbadrawy, A., and Karypis, G. User-specific feature-based similarity models for top-n recommendation of new items. *ACM Transactions on Intelligent Systems and Technology (TIST) 6*, 3 (2015), 1–20.

[21] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd* (1996), vol. 96, pp. 226–231.

[22] Fang, Y., and Si, L. Matrix co-factorization for recommendation with rich side information and implicit feedback. In *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems* (2011), pp. 65–69.

[23] Ferrari Dacrema, M., Boglio, S., Cremonesi, P., and Jannach, D. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems 39*, 2 (Jan. 2021), 1–49.

[24] Ferrari Dacrema, M., Cremonesi, P., and Jannach, D. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Sept. 2019), RecSys '19, ACM.

[25] Forgy, E. W. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *biometrics 21* (1965), 768–769.

[26] Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S., and Schmidt-Thieme, L. Learning attribute-to-feature mappings for cold-start recommendations. In *2010 IEEE international conference on data mining* (2010), IEEE, pp. 176–185.

[27] Gouvert, O., Oberlin, T., and Févotte, C. Matrix co-factorization for cold-start recommendation. In *19th International Society for Music Information Retrieval Conference (ISMIR 2018)* (2018), pp. 1–7.

[28] Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247* (2017).

[29] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis) 5*, 4 (2015), 1–19.

[30] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering, 2017.

[31] Hochreiter, S. Long short-term memory. *Neural Computation MIT-Press* (1997).

[32] Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6*, 02 (1998), 107–116.

[33] Hofmann, T. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS) 22*, 1 (2004), 89–115.

[34] Jeunen, O., Van Balen, J., and Goethals, B. Closed-form models for collaborative filtering with side-information. In *Proceedings of the 14th ACM Conference on Recommender Systems* (New York, NY, USA, 2020), RecSys '20, Association for Computing Machinery, p. 651–656.

[35] Kang, W.-C., and McAuley, J. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)* (2018), IEEE, pp. 197–206.

[36] Kim, J., and Park, H. Sparse nonnegative matrix factorization for clustering. Tech. rep., Georgia Institute of Technology, 2008.

[37] Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[38] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (2009), 30–37.

[39] Kuang, D., Ding, C., and Park, H. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM international conference on data mining* (2012), SIAM, pp. 106–117.

[40] Li, J., Wang, Y., and McAuley, J. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th international conference on web search and data mining* (2020), pp. 322–330.

[41] Lieberman, H., et al. Letizia: An agent that assists web browsing. *IJCAI (1) 1995* (1995), 924–929.

[42] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality, 2013.

[43] Mooney, R. J., and Roy, L. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries* (2000), pp. 195–204.

[44] Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. Mteb: Massive text embedding benchmark, 2023.

[45] Nguyen, J., and Zhu, M. Content-boosted matrix factorization techniques for recommender systems. *Statistical Analysis and Data Mining: The ASA Data Science Journal 6*, 4 (2013), 286–301.

[46] Nikolakopoulos, A. N., Ning, X., Desrosiers, C., and Karypis, G. Trust your neighbors: A comprehensive survey of neighborhood-based methods for recommender systems, 2021.

[47] Ning, X., and Karypis, G. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th international conference on data mining* (2011), IEEE, pp. 497–506.

[48] Ning, X., and Karypis, G. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the Sixth ACM Conference on Recommender Systems* (New York, NY, USA, 2012), RecSys '12, Association for Computing Machinery, p. 155–162.

[49] Paatero, P., and Tapper, U. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics 5*, 2 (1994), 111–126.

[50] Park, S.-T., and Chu, W. Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender systems* (2009), pp. 21–28.

[51] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[52] Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (2014), pp. 1532–1543.

[53] Puthiya Parambath, S. A., and Chawla, S. Simple and effective neural-free soft-cluster embeddings for item cold-start recommendations. *Data Mining and Knowledge Discovery 34* (2020), 1560–1588.

[54] Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks 12*, 1 (1999), 145–151.

[55] Reimers, N., and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[56] Rendle, S. Factorization machines. In *2010 IEEE International conference on data mining* (2010), IEEE, pp. 995–1000.

[57] Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[58] Rijsbergen, C. J. V. *Information Retrieval*, 2nd ed. Butterworth-Heinemann, USA, 1979.

[59] Salakhutdinov, R., Mnih, A., and Hinton, G. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning* (2007), pp. 791–798.

[60] Saveski, M., and Mantrach, A. Item cold-start recommendations: learning local collective embeddings. In *Proceedings of the 8th ACM Conference on Recommender systems* (2014), pp. 89–96.

[61] Schütze, H., Manning, C. D., and Raghavan, P. *Introduction to information retrieval*, vol. 39. Cambridge University Press Cambridge, 2008.

[62] Sharma, M., Zhou, J., Hu, J., and Karypis, G. Feature-based factorized bilinear similarity model for cold-start top-n item recommendation. In *Proceedings of the 2015 SIAM International Conference on Data Mining* (2015), SIAM, pp. 190–198.

[63] Singh, A. P., and Gordon, G. J. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), pp. 650–658.

[64] Sparck Jones, K. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation 28*, 1 (1972), 11–21.

[65] Steck, H. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference* (2019), pp. 3251–3257.

[66] Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management* (2019), pp. 1441–1450.

[67] Talha, M. M., Khan, H. U., Iqbal, S., Alghobiri, M., Iqbal, T., and Fayyaz, M. Deep learning in news recommender systems: A comprehensive survey, challenges and future trends. *Neurocomputing* (2023), 126881.

[68] Töscher, A., Jahrer, M., and Bell, R. M. The bigchaos solution to the netflix grand prize. *Netflix prize documentation* (2009), 1–52.

[69] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023.

[70] Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing 17* (2007), 395–416.

[71] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (2020), pp. 38–45.

[72] Xu, W., Liu, X., and Gong, Y. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (2003), pp. 267–273.

[73] Yahoo! Yahoo! movies user ratings and descriptive content information, version 1.0. `https://webscope.sandbox.yahoo.com/`, 2002.

[74] Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Yang, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao,

R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Liu, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., Guo, Z., and Fan, Z. Qwen2 technical report, 2024.

[75] Zeiler, M. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[76] Zhang, M., and Liu, K. Rethinking symmetric matrix factorization: A more general and better clustering perspective. In *2022 IEEE International Conference on Data Mining (ICDM)* (2022), IEEE, pp. 695–702.

[77] Zhang, S., Yao, L., Sun, A., and Tay, Y. Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR) 52*, 1 (2019), 1–38.

[78] Zhao, F., and Guo, Y. Learning discriminative recommendation systems with side information. In *IJCAI* (2017), vol. 2017, pp. 3469–3475.